REMOÇÃO HIERÁRQUICA DE GEOMETRIA POR OCLUSÃO EM SIMULAÇÕES EM TEMPO REAL

Vítor Manuel Rodrigues da Cunha

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização de Automação e Sistemas

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2009

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Vítor Manuel Rodrigues da Cunha, Nº 1080022, 1080022@isep.ipp.pt

Orientação científica: Professor João Miguel Queirós Magno Leitão, jml@isep.ipp.pt

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização de Automação e Sistemas

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

22 de Outubro de 2009

"…, the fastest polygon to render is the one never sent down the accelerator's pipeline."

in *Real Time Rendering (3rd Edition)*

Resumo

As aplicações de simulação gráfica em tempo real podem apresentar um fraco desempenho devido à elevada complexidade de ambientes virtuais de muito grandes dimensões. Um exemplo é a simulação de condução em ambientes urbanos onde se utilizam cenas extensas com densidade de geometria tipicamente muito elevada. Nesta perspectiva, o desenvolvimento de métodos que permitem a selecção apenas da geometria visível, eliminado toda a restante, é de importância crucial pois irá diminuir a quantidade de trabalho a realizar pelo *hardware* gráfico na geração da imagem final apresentada ao utilizador. Neste grupo de métodos, que determinam a visibilidade da geometria mediante um ponto de vista sobre a cena, existe um conjunto cujo propósito é o de determinar se um dado objecto no ambiente virtual se encontra ocluso por outro(s), logo, não visível. Este tipo de algoritmos de remoção por oclusão, em particular os que usam *Hardware Occlusion Queries*, tornou atractiva a determinação de visibilidade em tempo real sem necessidade de extensos cálculos em pré-processamento. Nesta dissertação é abordada a influência que a aplicação da remoção hierárquica por oclusão tem em simulações de ambientes virtuais complexos de muito grandes dimensões. Para comprovar de forma prática a análise efectuada, foi desenvolvida uma aplicação que usa as funcionalidades base que a API *OpenSceneGraph* disponibiliza para o *render* da cena e determinação de visibilidade. É proposto um conjunto de directivas a ter em consideração na aplicação da remoção por oclusão em estruturas hierárquicas de cenas com diferentes níveis de complexidade. Estas directivas são suportadas por resultados experimentais obtidos nas simulações realizadas. Propõe-se também um melhoramento da API *OpenSceneGraph* que permite resolver algumas das limitações do algoritmo de remoção por oclusão original disponibilizado pela API.

Palavras-Chave

simulação em tempo real, visibilidade, remoção por oclusão, teste de oclusão por *hardware*.

Abstract

Real time graphic simulations may show poor performance due to the high complexity of the very large virtual environments used. One example is the driving simulation in urban environments that typically uses large scenes with very high geometry density. Thus, the development of methods that select only the visible geometry, culling all the rest, is very important because it will reduce the amount of work performed by the graphics *hardware* in the rendering of the final image presented to the user. In this group of methods, that determine de visibility of the geometry based on the point of view into the scene, there are some whose purpose is to determine whether a given object in the virtual environment is occluded by other(s), thus, not visible. These type of occlusion culling algorithms, in particular those using the Hardware Occlusion Queries mechanism, made real time visibility computation attractive because it avoids extensive pre-processing calculations. This thesis deals with the influence in the overall performance that hierarchical occlusion culling algorithms have in the simulation of very large complex virtual environments. To prove this analysis in practice, an application was developed using the features that the *OpenSceneGraph* API provides for the rendering of the scene and to determine visibility. Several guidelines are proposed to be taken into account when applying occlusion culling methods to hierarchical structures of scenes with different levels of complexity. These directives are supported by experimental results obtained in the simulations. It's also proposed an improvement of the *OpenSceneGraph* API that allows solving some of the limitations of the original occlusion culling algorithm provided by the API.

Keywords

real-time rendering, visibility, occlusion culling, hardware occlusion query.

Résumé

Les applications de simulation graphique en temps réel ont un rendement médiocre en raison de la grande complexité des larges environnements virtuels. Un exemple est la simulation de conduite dans les milieux urbains où existent des endroits dont la géométrie représente une densité typiquement très élevée. En conséquence, le développement de méthodes pour sélectionner uniquement la géométrie visible, en éliminant toutes les autres, est crucial car elle diminuerait la quantité de travail nécessaire pour le *hardware* graphique pour générer l'image finale présentée à l'utilisateur. Dans cet ensemble de méthodes pour déterminer la visibilité de la géométrie par une vision de la scène, il existe un ensemble dont le but est de déterminer si un objet donné dans l'environnement virtuel est obstrué par d'autres, donc pas visible. Ce type d'algorithmes de suppression de l'occlusion, en particulier ceux utilisant des *Hardware Occlusion Queries*, est devenue attractive pour déterminer la visibilité en temps réel, sans nécessiter des calculs détaillés sur prétraitement. Dans cette dissertation, nous analysons l'influence que la demande de retrait de l'occlusion hiérarchique a sur la simulation d'environnements virtuels complexes et très grands. Pour passer de l'analyse à la pratique, nous avons développé une application en utilisant les fonctionnalités de l'API *OpenSceneGraph* que prévoit le *render* de la scène et détermine la visibilité. Elle propose un ensemble de directives à prendre en compte lors de la mise en œuvre la suppression de l'occlusion dans les structures hiérarchiques des scènes avec différents niveaux de complexité. Ces directives sont appuyées par des résultats expérimentaux obtenus dans les simulations. Il est également proposé une amélioration de l'API *OpenSceneGraph* qui permet de résoudre certaines des limitations de l'algorithme de suppression d'occlusion fourni par l'API d'origine.

Mots-clés

simulation en temps réel, visibilité, suppression de l'occlusion, test d'occlusion par *hardware*.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

2D – Bidimensional

3D – Tridimensional

ANSI – American National Standards Institute

API – Application Programming Interface

BSP – Binary Space Partitioning

CHC – Coherent Hierarchical Culling

cPLP – Conservative Prioritized-Layered Projection

CPU – Central Processing Unit

FPS – Frames Per Second

FSOC – Fast and Simple Occlusion Culling

GPU – Graphics Processing Unit

HOM – Hierarchical Occlusion Map

HZB – Hierarchical Z-Buffer

IOM – Incremental Occlusion Map

NOHC – Near Optimal Hierarchical Culling

OQ – Occlusion Query

OSG – OpenSceneGraph

PLP – Prioritized-Layered Projection

1. INTRODUÇÃO

No âmbito da computação gráfica, a crescente necessidade de modelos 3D mais detalhados em aplicações de simulação visual nem sempre é acompanhada pela evolução das capacidades do *hardware* gráfico. A elevada densidade de geometria acompanhada pelo elevado detalhe dos modelos que se estendem por vários quilómetros poderá comprometer a cadência de imagens e consequentemente degradar a experiência do utilizador de uma aplicação de simulação visual interactiva. Exemplos de modelos deste tipo são os ambientes urbanos (Figura 1).

Figura 1 – Exemplo de um ambiente virtual urbano (modelo *Town* do *Performer* [SGI]).

A densidade de geometria define a complexidade de uma cena 3D enquanto a extensão define a sua dimensão. A geração de imagens de uma cena a um ritmo que permita interactividade apresenta desafios em cenas com elevada complexidade e de muito grandes dimensões. Neste contexto, o desenvolvimento de técnicas que permitam diminuir a quantidade de geometria a desenhar em cada imagem ganha bastante relevância. Toda a geometria que é desenhada mas que não contribui para a imagem final representa um desperdício dos recursos de processamento ao nível do CPU e GPU. Uma forma de abordar a questão é remover toda a geometria não visível a partir do ponto de vista actual, ou seja, toda a geometria que não contribui para a imagem final. Neste conjunto de geometria pode ser incluída a que se encontra fora do volume de visualização (*view-frustum*), faces escondidas de objectos visíveis e geometria oclusa por outra que se encontre mais próxima do ponto de vista actual sobre a cena. Esta dissertação incide neste último conjunto de geometria. Especificamente, na forma como as técnicas usadas para remover geometria oclusa influenciam a simulação em tempo real de cenas 3D com elevada complexidade e de muito grandes dimensões.

1.1. CONTEXTUALIZAÇÃO

A simulação visual em tempo real refere-se à capacidade de uma aplicação criar imagens sintéticas suficientemente depressa num computador para que o utilizador possa interagir com um ambiente virtual. O processo de converter a informação 3D em píxeis que formam as imagens apresentadas ao utilizador é chamado de *rendering*. O processo de interacção e *rendering* acontece a um ritmo suficientemente rápido de forma que o utilizador não vê imagens individuais, em vez disso fica imerso num processo dinâmico. A taxa a que as imagens são mostradas é tipicamente medida em *frames* por segundo (fps).

Normalmente, os sistemas computacionais delegam as tarefas finais do *rendering* a um *hardware* dedicado, o acelerador gráfico. Este *hardware* gráfico contém um GPU que é responsável por gerar a imagem que mostra o conjunto de geometria que a aplicação, no CPU, decide desenhar. Para gerar uma imagem num GPU actual, a informação proveniente da aplicação é enviada por uma sequência de etapas consecutivas (*pipeline*). Assim, no caso de ser possível reduzir a quantidade de informação relativa a uma imagem na fase da aplicação, esta será gerada mais rapidamente pelo GPU.

Para conseguir taxas interactivas quando se simulam ambientes virtuais complexos, usualmente as aplicações computacionais, recorrem a técnicas que permitem remover geometria que não contribui para a imagem apresentada ao utilizador. Esta operação é chamada de *culling*. Do conjunto de toda a geometria que constitui um ambiente virtual (uma cena), para encontrar aquela que deve ser removida, podem ser aplicados algoritmos que determinam a visibilidade mediante o actual ponto de vista sobre a cena. Considere-se o exemplo de um percurso feito nas ruas de uma cena urbana. Do ponto de vista do utilizador, grande parte dos edifícios situados atrás dos edifícios visíveis encontram-se escondidos, bem como certamente o interior dos edifícios visíveis. Claramente, nesta situação o *rendering* de toda a geometria não visível é desnecessário. Assim, ao evitar o seu processamento não são ocupados recursos que poderão ser usados para gerar mais imagens por segundo.

Para lidar com o problema da visibilidade foram propostos ao longo dos anos vários algoritmos. Os algoritmos de remoção por oclusão, em particular, destinam-se a identificar geometria que apesar de presente no *view-frustum* não contribui para a imagem final por se encontrar oclusa. Este grupo de algoritmos evoluiu significativamente com a introdução e desenvolvimento de uma extensão do OpenGL, genericamente designada por *Hardware Occlusion Query*. Este mecanismo permite questionar o *hardware* gráfico relativamente à visibilidade de uma dada geometria face a outra(s) previamente desenhada(s). Apesar de na sua essência ser um mecanismo simples, o seu uso para obter um ganho significativo no desempenho em simulações em tempo real ainda apresenta desafios. A latência existente entre o teste por *hardware* à geometria e o retorno do resultado à aplicação pode influenciar negativamente o desempenho desta. Por exemplo, quando comparado com a aplicação do simples *view-frustum culling* [BIT04]. Diversos algoritmos que usam este mecanismo, analisados no Capítulo 4, propõem estratégias para minimizar este problema onde o denominador comum é a diminuição do número de testes à geometria realizados durante uma simulação. Com estas estratégias pretende-se que o algoritmo de remoção por oclusão ocupe a menor fracção possível do tempo de cálculo de uma *frame*. Os critérios opostos que se aplicam à remoção por oclusão são a velocidade do algoritmo em relação à precisão da classificação de visibilidade resultante da sua aplicação.

1.2. OBJECTIVOS

A determinação de visibilidade pode ser definida como: dada uma cena e um ponto de vista, rapidamente determinar o conjunto de geometria visível, *i.e.*, o conjunto de objectos que potencialmente contribuem para a imagem final. Estes objectos são então enviados para a *pipeline* gráfica para processamento e a restante geometria da cena excluída.

A tarefa de encontrar o conjunto de geometria que contribui para a imagem pode incluir a remoção de objectos oclusos em relação ao ponto de vista actual, *i.e.*, a remoção por oclusão. Uma estratégia promissora parece ser a exploração da organização espacial hierárquica de cenas no processo de remoção deste tipo de geometria. Em cenas organizadas hierarquicamente (*e.g.*, grafo de cena), a aplicação do mecanismo das *Hardware Occlusion Queries* deverá levar a uma diminuição no número de operações necessárias para remover toda a geometria oclusa. Se um nível superior da hierarquia for determinado como ocluso por outra geometria mais próxima do ponto de vista, ao ser removido de posteriores processamentos, também todos os níveis hierárquicos inferiores a si serão removidos (árvore filha). Nesta situação são expectáveis ganhos devidos à geometria não processada e testes não realizados.

O principal objectivo a que este trabalho se propõe é analisar a influência que a remoção por oclusão tem quando aplicada a diferentes níveis hierárquicos de uma cena com elevada complexidade de geometria e de muito grandes dimensões. Diversas simulações serão realizadas para identificar as vantagens e fraquezas da remoção deste tipo de geometria quando aplicada a diferentes níveis hierárquicos e combinações destes. O impacto que esta forma de remover geometria oclusa tem no desempenho da aplicação que simula a cena em tempo real também será analisado. Este estudo surge da ausência de informação relativa ao objectivo a que se propõe esta dissertação, em particular, por parte dos algoritmos mais recentemente apresentados que abordam o problema da remoção por oclusão assistida por *hardware*. Um objectivo adicional deste trabalho é a análise da influência que a aplicação do princípio da coerência temporal tem quando aplicado a geometria da cena em diferentes situações de visibilidade.

1.3. CALENDARIZAÇÃO

O trabalho realizado no âmbito da presente dissertação foi realizado de acordo com as tarefas listadas na Figura 2. Numa primeira fase começou-se por pesquisar documentação relevante na área e estudar a API que seria usada na aplicação a implementar, necessária a tarefas posteriores. O desenvolvimento da aplicação a usar nas simulações em conjunto com a criação das cenas 3D constituíram as tarefas que precederam a realização do primeiro conjunto de simulações. Estas visaram o estudo da influência da remoção hierárquica de geometria oclusa no desempenho da aplicação. Em seguida, e com os conhecimentos adquiridos na primeira fase do trabalho, realizou-se um segundo conjunto de simulações com o objectivo de analisar o impacto de diferentes técnicas de actualização da informação de visibilidade da geometria presente na cena 3D. As tarefas finais consistiram na redacção da presente dissertação e na elaboração de um artigo que foi submetido ao 17º Encontro Português de Computação Gráfica, no qual foi aceite.

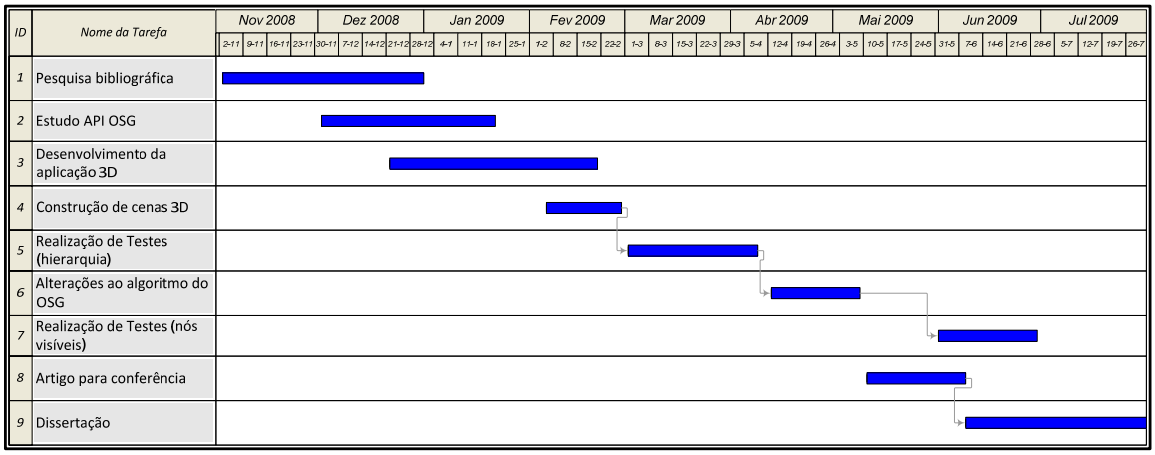


Figura 2 - Calendarização de tarefas.

1.4. ORGANIZAÇÃO DO DOCUMENTO

A estrutura da dissertação encontra-se dividida em 7 capítulos. O primeiro capítulo, o presente, introduz e contextualiza o trabalho em questão, apresenta os objectivos e a calendarização das tarefas realizadas. O segundo capítulo dedica-se a introduzir, de forma sucinta, diversos conceitos fundamentais associados à determinação de visibilidade em simulações em tempo real. No capítulo seguinte, o terceiro, é dada uma visão geral sobre algumas técnicas de remoção por oclusão assistidas pelo *hardware* gráfico, com especial

incidência na que se baseia no mecanismo de *Hardware Occlusion Queries* por OpenGL. Duas das técnicas descritas não são actualmente usadas mas a sua contribuição para o campo da remoção hierárquica por oclusão é bastante relevante. O Capítulo 4 apresenta e analisa alguns dos mais importantes algoritmos de remoção por oclusão baseados no mecanismo de OpenGL apresentados nos mais recentes anos. No Capítulo 5, é apresentado o projecto OpenSceneGraph cuja API é usada na aplicação desenvolvida para este trabalho e que fornece as funcionalidades base de *rendering* e organização hierárquica da informação espacial. Neste capítulo é também analisado em detalhe o algoritmo de remoção por oclusão implementado por esta API. No Capítulo 6 é analisada a influência da hierarquia na remoção por oclusão, descritas as cenas 3D criadas e apresentada a aplicação desenvolvida. São também apresentados e analisados os resultados experimentais obtidos nas diferentes simulações. No final do capítulo é testado um método de remoção por oclusão alternativo ao implementado pelo OpenSceneGraph. Por fim, no Capítulo 7, são apresentadas as conclusões finais e possíveis direcções a tomar em desenvolvimentos futuros.

2. CONCEITOS PRELIMINARES

Este capítulo fornece uma introdução a conceitos associados aos capítulos seguintes e ao âmbito geral desta dissertação. Estes incluem a organização de cenas tridimensionais em estruturas de dados e uma perspectiva sobre os métodos mais comuns de determinação de visibilidade usados em simulações visuais em tempo real.

2.1. ORGANIZAÇÃO DE CENAS TRIDIMENSIONAIS

No contexto da computação gráfica, uma cena consiste num conjunto de objectos que podem ser criados através de um *software* de modelação ou resultantes de um qualquer processo de aquisição de informação 3D. Os objectos consistem num conjunto de faces, geralmente triângulos (as palavras primitiva e polígono são também usadas para identificar estes blocos elementares de construção). Assumindo que uma face é representada como um triângulo, esta pode ser completamente definida através de três pontos no espaço tridimensional, os vértices. Na realidade, mais informação é normalmente guardada, tal como a cor da face, a normal à face, coordenadas de textura, entre outras informações necessárias para visualizar o objecto. Todos os objectos que constituem uma cena são colectivamente denominados de geometria da cena.

A geometria de uma cena pode ser classificada de estática ou dinâmica. Toda a geometria que não pode ser sujeita a qualquer alteração durante uma simulação, tal como estradas e edifícios numa cena urbana, é denominada de estática. A geometria dinâmica é aquela que pode mudar de forma ou mover-se através da cena, por exemplo um automóvel a deslocar-se pelas ruas de uma cidade.

Para organizar e gerir toda a informação que compõe uma cena 3D são usadas, tipicamente, estruturas de dados hierárquicas. A principal motivação para usar hierarquia é o custo de processamento em algoritmos que necessitam de percorrer a estrutura de dados que poderá passar de $O(n)$ para $O(\log n)$. Exemplos são algoritmos de remoção por oclusão, detecção de colisões, *ray tracing*, etc. As estruturas hierárquicas são particularmente adequadas à determinação de visibilidade porque quando uma porção da cena é determinada como não visível, toda a geometria a ela associada em níveis hierárquicos inferiores, pode também ser removida de posteriores processamentos. Convém no entanto salientar que a maioria das estruturas de dados hierárquicas possui um elevado custo de construção pelo que são tipicamente criadas numa etapa de pré-processamento. A actualização das estruturas hierárquicas durante a execução é viável de uma forma geral, no entanto, algumas estruturas são mais adequadas para cenas dinâmicas do que outras.

Alguns tipos de estruturas de dados espaciais são as Hierarquias de Volumes Envolventes [RUB80], *octrees* [GLA84] e árvores BSP [BEN75]. As *octrees* e árvores BSP são estruturas de dados baseadas na subdivisão do espaço. Isto significa que todo o espaço da cena é subdividido e incluído na estrutura de dados. Por exemplo, a união dos espaços representados nos nós folha da árvore deverá perfazer o espaço total da cena. As árvores BSP são irregulares, o que significa que o espaço pode ser subdividido de forma mais arbitrária. As *octrees* são regulares, o que significa que a divisão do espaço é realizada de uma forma uniforme. Apesar de mais restritivas, esta uniformidade pode ser uma fonte de eficiência. As Hierarquias de Volumes Envolventes, por outro lado, não são uma estrutura de subdivisão do espaço. Estas apenas incluem as regiões do espaço que rodeiam objectos geométricos. Desta forma não necessitam de incluir todo o espaço da cena na estrutura. Por fim serão apresentados os grafos de cenas que são estruturas de dados de mais alto nível que também não usam uma subdivisão do espaço estrita.

2.1.1. HIERARQUIA DE VOLUMES ENVOLVENTES

Um volume envolvente é um volume que engloba um conjunto de primitivas 3D. Para que o uso de volumes envolventes seja vantajoso, a sua geometria deverá ser bastante mais simples que a geometria por si envolvida. Desta forma, realizar operações com estes volumes será mais rápido do que seria com os objectos em si. Exemplos de volumes envolventes são esferas, *axis-aligned bounding boxes*, *oriented bounding boxes* e *k*-DOPs (ver [AKE08] para uma explicação aprofundada). Os volumes envolventes são usados em variados tipos de cálculos e pesquisas com o objectivo de acelerar a geração da imagem. No entanto, não contribuem visualmente para a imagem final apresentada ao utilizador.

Existente já em 1980 [RUB80], a Hierarquia de Volumes Envolventes consiste numa estrutura em árvore com raiz, folhas e nós internos. O nó de mais alto nível é o nó raiz, que não tem pais. Um nó folha contém a geometria a ser desenhada e não tem filhos. Em contraste, um nó interno tem apontadores para os seus filhos. Cada nó, incluindo os nós folha, representa na árvore um volume envolvente que engloba a geometria de toda a árvore filha a si ligada, daí o nome de Hierarquia de Volumes Envolventes. Isto significa que o nó raiz tem um volume envolvente que contém toda a cena, um exemplo é apresentado na Figura 3.

Este tipo de estrutura também pode ser usado para cenas com objectos dinâmicos. Neste caso, uma possível estratégia é remover o objecto do volume envolvente do seu pai sempre que se move para fora deste, o que implica recalcular o volume envolvente do pai. A sua nova posição na hierarquia é encontrada recursivamente a partir do nó raiz.

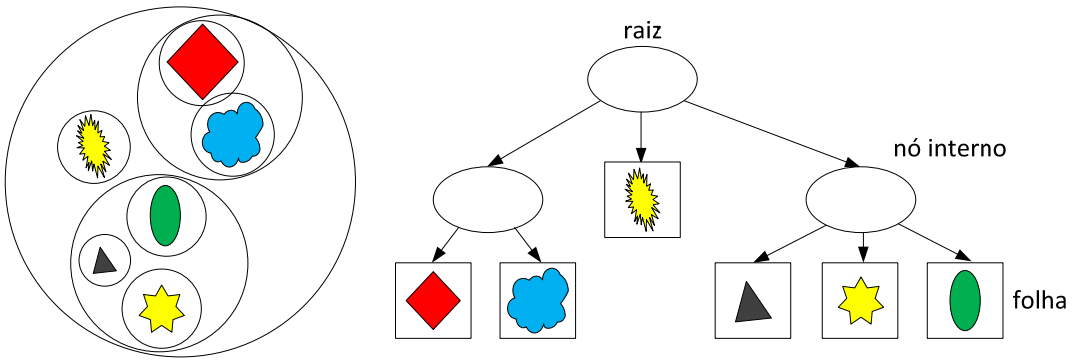


Figura 3 – Hierarquia de Volumes Envolventes: exemplo com esferas como volumes envolventes.

2.1.2. SUBDIVISÃO DO ESPAÇO

Octree

Para construir uma *octree* [GLA84] toda a cena é inicialmente encapsulada por uma caixa envolvente de lados iguais alinhada com os eixos (*axis-aligned bounding box*). A dimensão da caixa é determinada pela extensão máxima no espaço que a cena ocupa em qualquer um dos eixos tridimensionais. A caixa envolvente é então dividida ao longo dos três eixos com o ponto de divisão no centro geométrico do volume. Esta operação cria oito novos volumes, daí o nome *octree*. O processo de subdivisão continua recursivamente até que determinado critério de paragem tenha sido atingido. A subdivisão pode parar quando tenha sido atingida uma determinada profundidade no processo recursivo ou quando o número de primitivas numa caixa perfaça um valor predefinido.

A estrutura resultante deste processo aplicado em 2D denomina-se de *quadtree*. A Figura 4 ilustra o processo de criação de uma destas estruturas aplicado a uma cena exemplo. Da esquerda para a direita, a figura mostra as sucessivas divisões do espaço que terminam, neste caso, quando cada célula se encontra vazia ou contém um objecto.

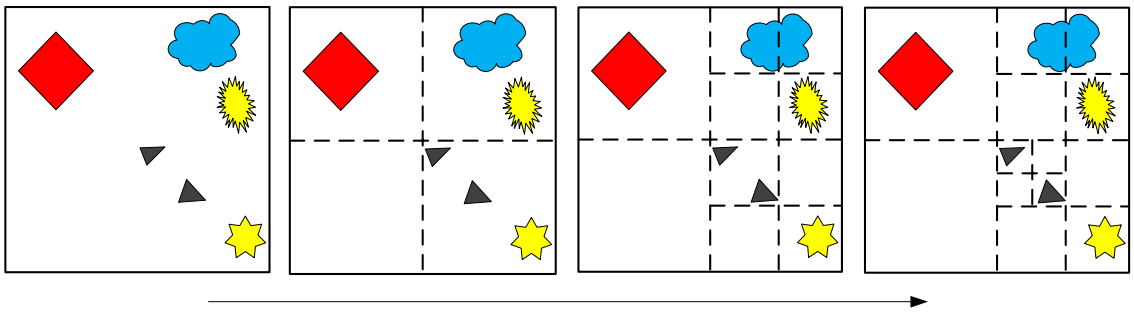


Figura 4 - Processo de criação de uma *quadtree*.

Este esquema de divisão do espaço em cubos não se adapta a cenas predominantemente planas, como por exemplo uma cidade. Também não é apropriado para cenas dinâmicas dado que um objecto em movimento alteraria o conteúdo das caixas o que poderia levar a que houvesse a necessidade de fundir caixas quase vazias ou dividir caixas que se tenham tornado demasiado ocupadas. Estas operações são potencialmente custosas e devem ser evitadas. No entanto, sendo a *octree* uma estrutura hierárquica, proporciona a seguinte grande vantagem. No caso de um nó num nível superior da árvore for

determinado como não visível, todos os seus nós filhos podem ser removidos. Este é o princípio fundamental que torna as estruturas de dados hierárquicas adequadas à determinação de visibilidade.

Árvore BSP

Uma árvore BSP (*Binary Space Partitioning*) [BEN75] é criada de forma semelhante à *octree*. Tal como numa *octree* o processo inicia-se com a determinação da caixa envolvente mínima alinhada com os eixos que envolve toda a cena, mas ao contrário da *octree* esta não necessita de ser cúbica. A principal característica da árvore BSP é ser binária. Em cada subdivisão o espaço é dividido em dois através de um plano, o que origina que cada nó terá sempre ou zero ou dois filhos. Este processo é aplicado recursivamente. Enquanto na *octree* a árvore é construída dividindo de forma regular o espaço (todos os filhos de um dado nó têm o mesmo tamanho), na árvore BSP, o local onde é realizada a divisão pode ser escolhido de acordo com uma métrica. Esta pode ser, por exemplo, igual número de objectos nos dois subespaços ou o menor número possível de objectos subdivididos. A aplicação recursiva do método termina quando é atingido um critério predefinido. Na Figura 5 é ilustrada a sequência de divisões BSP aplicada a uma cena 2D exemplo. A numeração indica a ordem pela qual os planos de divisão foram aplicados e a correspondente estrutura hierárquica obtida.

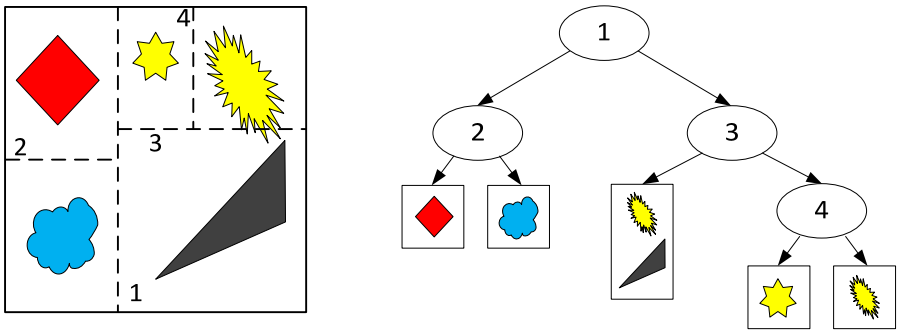


Figura 5 - Cena 2D particionada por uma árvore BSP *axis-aligned*.

Existem duas variantes deste método, a anteriormente descrita, denominada de *axis-aligned* BSP (ou *kD-tree*) que é a variante mais usada, e a *polygon-aligned* BSP. Ambas implicam processos de construção morosos devido à dificuldade de encontrar "bons" planos para dividir a cena, devido a isto, são usualmente construídas em pré-

processamento e guardadas para futuro uso. Assim, são mais indicadas para cenas estáticas sendo usadas em vários algoritmos modernos [AIL04, BIT04]. Além disso, permitem com facilidade ordenar o conteúdo geométrico da árvore da frente para trás relativamente a qualquer ponto de vista, característica importante para a determinação de visibilidade.

2.1.3. GRAFOS DE CENA

Os grafos de cena [AKE08] são estruturas de dados hierárquicas em árvore que organizam informação de cenas 3D. Em contraste com as Hierarquias de Volumes Envolventes, *octrees* e árvores BSP que estão vocacionadas apenas para guardar geometria, os grafos de cena são enriquecidos com texturas, transformações (*e.g.*, rotação, translação), níveis de detalhe, estados (*e.g.*, propriedades de materiais), fontes de luz, entre outros. Esta característica torna os grafos de cena mais versáteis e indicados para manipular, guardar e organizar informação de uma cena 3D.

Um grafo de cena é encabeçado por um nó raiz que representa a base da estrutura em árvore da cena. Ligados a este, podem existir nós grupo internos que organizam a geometria e o estado de *render* que controla a sua aparência. A geometria que constitui os objectos da cena encontra-se em nós folha, ligados a nós grupo.

Quando vários nós apontam para um mesmo nó filho a estrutura em árvore é chamada de Grafo Direccionado Acíclico [AKE08]. O termo acíclico significa que o grafo não pode ter nenhum ciclo fechado. Por direccionado entende-se que dois nós estão ligados por uma determinada ordem, *i.e.*, de pai para filho.

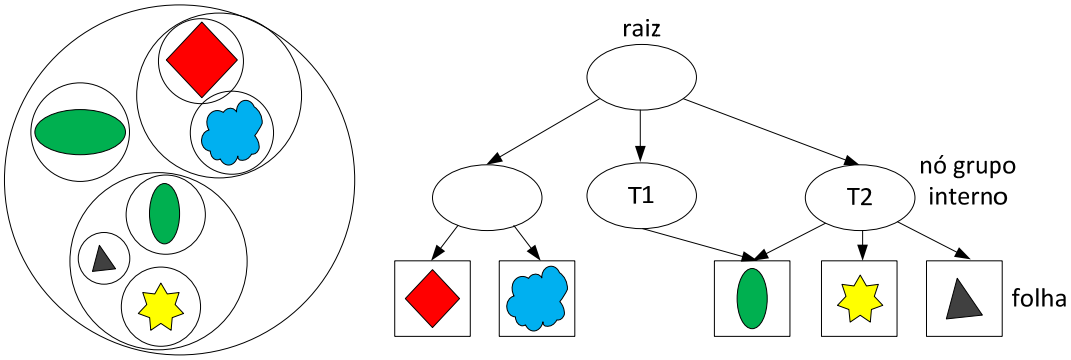


Figura 6 - Grafo de cena com diferentes transformações aplicadas a nós internos.

Os grafos de cena são frequentemente grafos acíclicos para permitirem instanciação, ou seja, fazer várias cópias de um objecto sem a necessidade de replicar a sua geometria, este processo permite poupar recursos de *hardware*. A Figura 6 apresenta um grafo de cena abstracto que reflecte a estrutura em grafo descrita, neste caso, o nó folha com a figura oval é desenhado duas vezes devido a ter duas transformações aplicadas (T1 e T2).

Para ilustrar o conceito de estrutura hierárquica considere-se a animação de um objecto. Num grafo de cena esta tarefa pode ser implementada aplicando transformações a nós internos da árvore que depois se propagam por todo o conteúdo da árvore filha desse nó. Como uma transformação pode ser colocada em qualquer nó interno, a animação hierárquica é assim conseguida. Por exemplo, as rodas de um automóvel podem rodar e o automóvel andar em frente como um todo. Na Figura 7 a transformação aplicada pelo nó em evidência permite deslocar (por herança) toda a geometria com uma única operação. As transformações aplicadas unicamente aos nós folha permitem, por exemplo, ajustar as rodas da frente para reflectirem as mudanças de direcção do automóvel.

De uma forma simples, para gerar uma imagem a partir da informação guardada no grafo de cena a árvore é varrida recursivamente em profundidade para actualizar os nós e recolher informação necessária ao seu *render*. Se um nó for visível e possuir geometria é enviado para o subsistema responsável pelo *render* da geometria. Usualmente estas tarefas (actualizar, determinar a visibilidade e o *rendering*) são realizadas de forma intercalada para distribuir a carga de processamento entre o CPU e o GPU.

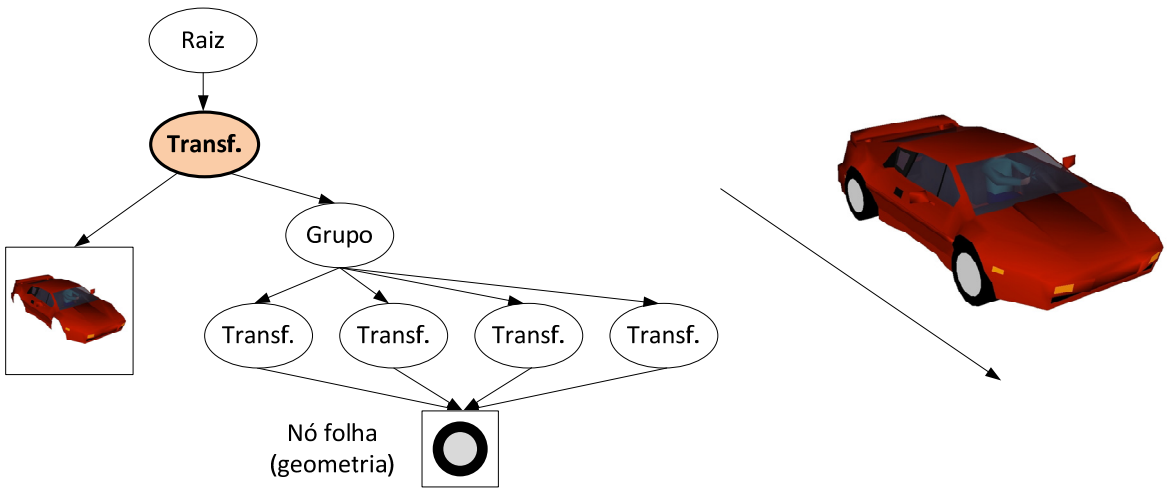


Figura 7 – Animação conseguida por herança num grafo de cena.

A eficiência computacional e versatilidade dos grafos de cena levam a que a maioria das aplicações gráficas empregue algum tipo de grafo de cena. Para desenvolver estas aplicações existem várias bibliotecas: Open Inventor [WER94], IRIS Performer [ROH94], OpenSG [OpSG], OpenSceneGraph [OSG], entre muitas outras. Apesar das diferenças das diversas bibliotecas (*e.g.*, linguagem de programação, estruturas de dados, arquitectura de *software*, técnicas de varrimento) todas usam OpenGL [WOO97] ou Direct3D [D3D] para o *rendering* da geometria. No desenvolvimento da aplicação de suporte a esta dissertação foi usado o grafo de cena disponibilizado pelo OpenSceneGraph que será abordado com maior detalhe no Capítulo 5.

2.2. *RENDERING* DE GRAFOS DE CENA

O processamento mais comum de um grafo de cena é realizado percorrendo o grafo em profundidade e enviando a informação de estado e geometria recolhida para o *hardware* gráfico através de comandos OpenGL ou Direct3D. Este processo é usualmente feito em todas as *frames*. Para actualizar dinamicamente a geometria, remover objectos não visíveis, ordenar e realizar um *rendering* eficiente, entre outras tarefas, existem tipicamente três etapas no *rendering* de grafos de cena:

- Actualização: nesta etapa é realizada a actualização do estado e da geometria, o que permite cenas dinâmicas. As actualizações são realizadas directamente nos nós ou através do mecanismo de *Callbacks* onde são atribuídas funções aos nós do grafo de cena a actualizar. A aplicação usa a etapa de actualização para modificar, por exemplo, a posição de um avião num simulador de voo ou para permitir modificações devido à interacção com o utilizador através de mecanismos de entrada.

- Remoção (ou *Culling*): depois da etapa de Actualização é realizada a Remoção. Nesta etapa a árvore de objectos é percorrida para seleccionar que partes da cena são visíveis na *frame* actual. Desta forma, é realizada a determinação de visibilidade da cena 3D mediante o ponto de vista actualizado na etapa anterior. Se um objecto for considerado visível é adicionado ao conjunto de objectos a desenhar durante o processo de visualização. Desta etapa resulta uma estrutura

de dados que pode ser uma lista onde os objectos são ordenados mediante critérios como transparência e profundidade (distância ao ponto de vista).

- Desenho (ou *Draw*): esta etapa consiste no processo de visualização que percorre a lista de geometria criada durante a etapa de Remoção e realiza chamadas à API gráfica de baixo nível para desenhar a geometria. É nesta etapa que a geometria é enviada para o *hardware* gráfico.

Tipicamente, estas três etapas são realizadas uma vez por *frame*. No entanto, em algumas situações é necessário gerar várias vistas simultâneas sobre a mesma cena (*e.g.*, visualização estéreo, vários dispositivos de visualização). Nestas situações, a etapa de Actualização é executada uma vez por *frame*, enquanto as etapas de Remoção e Desenho são executadas uma vez por cada *frame* e visualização.

2.3. VISIBILIDADE EM CENAS TRIDIMENSIONAIS

Durante a etapa de Remoção do processo de *rendering* de grafos de cena é determinada a visibilidade da cena 3D. Em simulações em tempo real as restrições de tempo são um factor importante, o sistema não deverá utilizar demasiado tempo no cálculo da visibilidade de forma a manter uma cadência de imagens que permita uma experiência interactiva. A principal ideia subjacente aos algoritmos de visibilidade é de encontrar rapidamente uma estimativa das partes da cena que se encontram visíveis para poder remover as restantes.

Usando como exemplo um simulador de condução num ambiente urbano, o desafio de determinar a visibilidade consiste em identificar os objectos da cena que não são visíveis, ou por se encontrarem fora do campo de visão do utilizador ou por se encontrarem escondidos por objectos (*e.g.*, edifícios) mais próximos do ponto de vista actual. Em cenas com elevada complexidade em profundidade, este último grupo de objectos assume um papel ainda mais importante, pois no caso de não serem identificados como oclusos serão enviados desnecessariamente para o *hardware* gráfico e assim consumir recursos importantes, especialmente em simulações em tempo real.

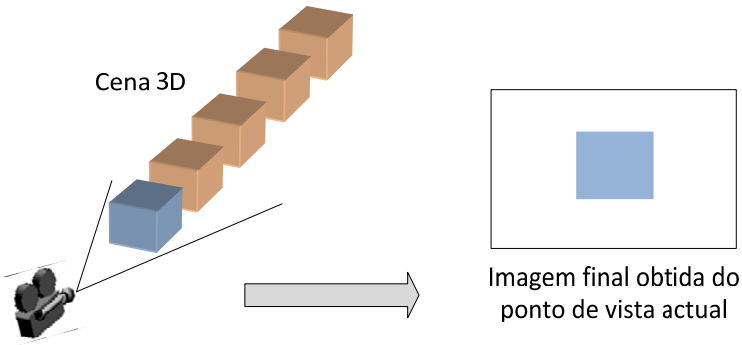


Figura 8 - Exemplo de uma cena com elevada complexidade em profundidade.

A cena 3D ilustrada na Figura 8, apesar de artificial, demonstra uma situação onde é desejável evitar o processamento da geometria que não contribui para a imagem final. Se os cubos forem desenhados de trás para a frente, relativamente à câmara, a mesma zona do ecrã será escrita várias vezes apesar de na imagem final apenas ser visualizado o cubo mais próximo do ponto de vista. Se o processamento for realizado da frente para trás toda a geometria é enviada para o *hardware* gráfico, apesar de na imagem apenas ser visível o cubo mais próximo da câmara. Neste caso, o mecanismo de visibilidade exacta do *hardware* gráfico (apresentado no Capítulo 3) faz com que apenas o cubo mais próximo do ponto de vista seja desenhado. No entanto, se com a mesma ordem de processamento for possível detectar objectos oclusos na etapa de Remoção, apenas o cubo mais próximo do ponto de vista será enviado para o *hardware* e desenhado.

O algoritmo de visibilidade ideal enviaria apenas o conjunto exacto de primitivas visíveis para *rendering*. Uma estrutura de dados que permite a remoção ideal é o grafo de aspecto [GIG88]. A partir deste grafo, o conjunto exacto de primitivas para qualquer ponto de vista pode ser extraído. No entanto, apesar de teoricamente possível, na prática a complexidade na sua criação pode chegar a valores na ordem de $O(n^9)$ [COH03].

2.3.1. TÉCNICAS DE DETERMINAÇÃO DE VISIBILIDADE

A determinação de visibilidade pode ser encarada como uma *pipeline* [AIL00] onde diferentes técnicas são aplicadas para realizar diferentes tipos de remoção de geometria. A presente secção descreve algumas dessas técnicas.

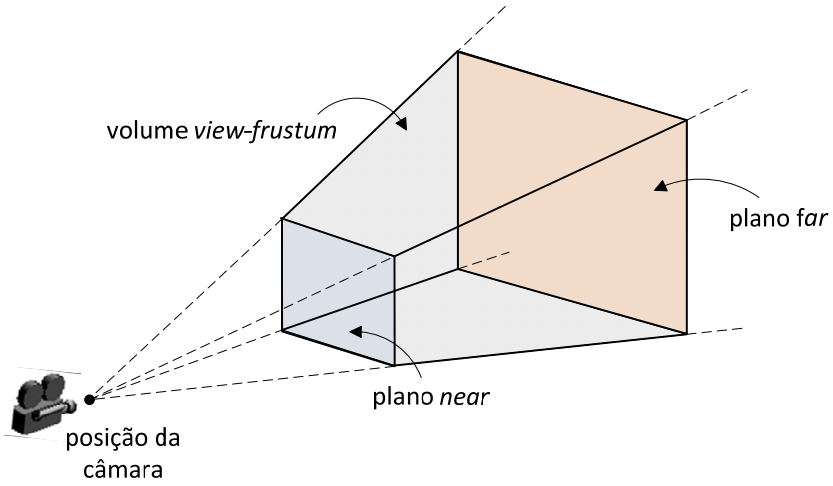


Figura 9 - *View-frustum*.

View-frustum culling

Por *view-frustum* entende-se o volume de cena visível pelo utilizador. Este volume, devido à projecção em perspectiva, assume a forma de uma pirâmide. Para definir completamente o *view-frustum* é necessário considerar mais dois planos que irão formar um volume limitado por seis planos (Figura 9). Quatro desses planos podem ser encarados como uma superfície que se inicia na câmara e se estende até infinito tendo como limitação o ecrã de computador (pirâmide). Os restantes dois planos são ortogonais à direcção de visualização e são designados *near* e *far*. O plano *near* funciona como uma "janela" através da qual o utilizador vê a cena, geralmente encontra-se bastante perto do ponto de vista. A distância a que se encontra o plano *far* depende do tipo de cena. Em cenas com elevada densidade geométrica esta pode ser relativamente curta enquanto em extensas cenas exteriores deverá ser elevada.

O *view-frustum culling* consiste na remoção do processo de *rendering* das partes da cena que se encontram fora do *view-frustum*. Nesta dissertação optou-se por referir a remoção do volume de visualização pela sua designação anglo-saxónica devido a ser um termo amplamente conhecido no seio da computação gráfica.

No processo de determinação de visibilidade são testados os volumes envolventes dos objectos que se podem encontrar em três situações: dentro, fora ou a intersectar o *view-frustum*. Se o volume envolvente se encontrar fora, a geometria por este englobada pode ser omitida de posteriores processamentos. Como estes cálculos são realizadas no CPU,

isto significa que a geometria contida no volume envolvente não necessita de ser enviada para o *hardware* gráfico. Se o volume envolvente se encontrar a intersectar, a geometria é considerada visível ou então mais testes serão realizados no caso de representar um nó interno de uma estrutura hierárquica. As três situações encontram-se ilustradas na Figura 10, onde a geometria removida de posteriores processamentos é representada a tracejado.

Em cenas de grandes dimensões ou em pontos de vista específicos de uma cena, apenas uma fracção da cena poderá contribuir para a imagem final, num dado momento. Nestes casos, a aplicação do *view-frustum culling* permite determinar essa fracção da cena que necessita de ser enviada para *rendering*. Assim, é de esperar um enorme ganho no desempenho da aplicação com a aplicação desta técnica de determinação de visibilidade.

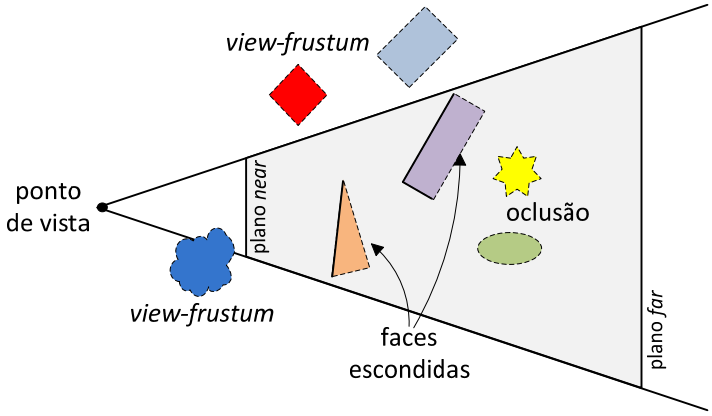


Figura 10 - *View-frustum culling*, remoção por oclusão e remoção de faces escondidas.

Remoção por oclusão

A remoção por oclusão (*occlusion culling*) refere-se à identificação de objectos que não contribuem para a imagem final por se encontrarem obstruídos da vista por outra geometria mais próxima do ponto de vista actual (Figura 10). Estes objectos oclusos, quando identificados podem ser removidos do processo de *rendering*. Uma remoção por oclusão óptima implicaria que apenas os objectos visíveis que se encontram mais próximo do ponto de vista seriam desenhados. Desta forma, a remoção de geometria oclusa é fulcral para se conseguir baixar a complexidade em profundidade. Este é um problema complexo que é abordado em detalhe no Capítulo 3.

Remoção de faces escondidas

Numa cena 3D com objectos opacos é fácil de perceber que as faces não visíveis dos objectos não necessitam de ser desenhadas pois não contribuem para imagem final. A remoção de faces escondidas (*backface culling*) prende-se com a remoção de polígonos de objectos, em que a normal do polígono aponte para longe do utilizador. Isto significa que não são visíveis do actual ponto de vista podendo ser removidas.

Para determinar se uma face se encontra escondida são necessários dois vectores. Um é o vector do ponto vista à face a testar, o outro é a normal à face. Este último pode ser facilmente encontrado através dos vértices da face [AKE08]. Se o ângulo entre estes dois vectores for superior a 90° a face não se encontra orientada para o ponto de vista e assim pode deixar de ser desenhada. Na prática, basta estudar o sinal do produto escalar entre os dois vectores, se este for inferior a zero a face encontra-se escondida. A Figura 10 mostra a aplicação da remoção de faces escondidas no contexto das restantes técnicas de visibilidade analisadas. A Figura 11 ilustra o teste a diversas faces de objectos visíveis. Neste caso, as faces *B* e *C* são visíveis enquanto a face *A*, com um ângulo superior a 90º, é identificada como escondida.

Outra forma de identificar as faces escondidas de objectos é através da normal da face depois de projectada no espaço bidimensional do ecrã. Esta normal apenas poderá ter dois sentidos. Assim, se o eixo negativo de *Z* estiver direccionado para o interior da cena, a face estará escondida se a sua normal apontar no sentido negativo de *Z*.

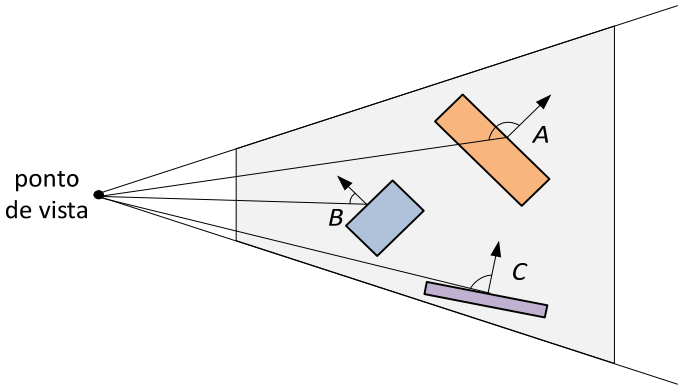


Figura 11 - Teste para determinar faces escondidas de objectos opacos visíveis.

Este tipo de remoção é suportado pela generalidade do actual *hardware* gráfico. Esta operação exige que os polígonos candidatos a serem removidos tenham que ser enviados para o GPU e parcialmente processados antes do teste. A remoção de faces escondidas por *hardware* processa os polígonos linearmente, no entanto, também existem técnicas que permitem remover faces escondidas hierarquicamente [KUM96].

2.3.2. VISIBILIDADE CONSERVADORA E APROXIMADA

A imagem final gerada para um ponto de vista sobre a cena é correcta desde que todas as primitivas que para ela contribuem sejam processadas. No entanto, se objectos da cena se encontrarem oclusos por outros não faz sentido desperdiçar recursos computacionais a processar estes objectos. Os algoritmos de visibilidade conservadores tentam encontrar objectos que não contribuem para a imagem final possibilitando a remoção dessa geometria sem comprometer a correcção da imagem final apresentada ao utilizador. Estes algoritmos são complexos e é improvável que um único algoritmo resolva todos os aspectos relacionados com a remoção de geometria não visível. A *taxa de conservação* de um algoritmo pode ser determinada pela métrica [AIL00]:

$$taxa\ de\ conservação = \frac{R}{V}$$

Onde:

- *R*, é a quantidade de geometria visível identificada pelo algoritmo de visibilidade;
- *V*, representa a quantidade real de geometria visível. Esta quantidade pode ser obtida, por exemplo, fazendo o *render* de cada polígono com uma cor diferente e depois ler o *frame buffer* para contabilizar o número de polígonos que aparecem na imagem final [HIL02].

O termo "conjunto potencialmente visível" é usado para designar o conjunto de geometria identificada por um algoritmo de visibilidade conservador. Estes algoritmos para garantirem uma imagem final correcta sobrestimam a quantidade geometria visível pelo que o conjunto potencialmente visível irá conter geometria visível e alguma não visível.

Existem também algoritmos que optam por uma abordagem mais agressiva. Estes algoritmos de visibilidade aproximada subestimam o conjunto potencialmente visível, desta forma sacrificam a correcção da imagem por uma melhoria no desempenho. Um exemplo deste tipo de algoritmos é descrito em [KLO00]. A determinação da visibilidade aproximada pode ser usada, por exemplo, em aplicações que necessitam de manter em todos os instantes a mesma capacidade de resposta, mesmo que isso signifique abdicar de algum realismo da cena. O erro entre a geometria realmente visível e a considerada visível por algoritmos de visibilidade aproximada pode ser definido com base na *taxa de conservação*:

$$erro = 1 - taxa\ de\ conservação = \left(1 - \frac{R}{V}\right)$$

2.3.3. COERÊNCIA

A exploração da coerência é um factor bastante importante para um algoritmo de visibilidade. Três tipos de coerência são de especial interesse: espaço dos objectos, espaço das imagens e temporal (Figura 12). Outros tipos de coerência encontram-se identificados em [GRO95].

A coerência no espaço dos objectos (espacial) é baseada nas relações entre os objectos ou partes de objectos que se encontram próximos no espaço 3D da cena. Assim, um cálculo realizado para um grupo de objectos pode ser aplicado aos objectos individualmente. Por exemplo, se o volume envolvente de um objecto é determinado como invisível, o objecto por si contido não necessita de ser processado.

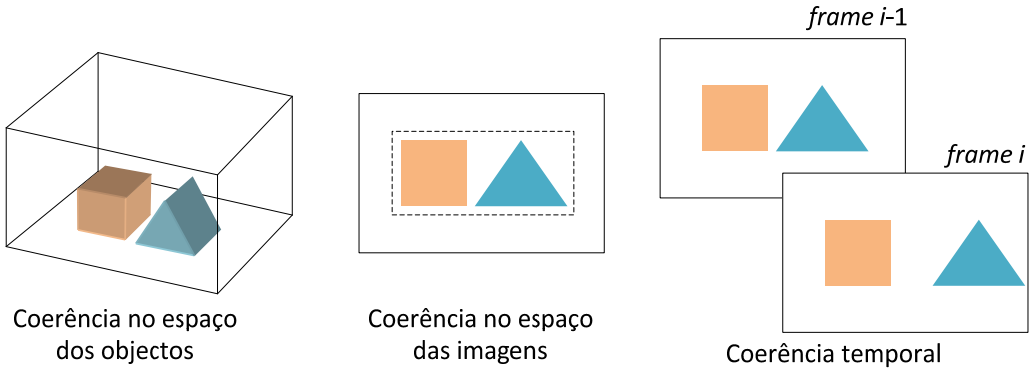


Figura 12 - Três formas de coerência que um algoritmo de visibilidade pode explorar.

A coerência no espaço das imagens é análoga à coerência espacial mas dependente do ponto de vista. Na imagem criada de uma cena 3D, píxeis que se encontram próximos muitas vezes representam o mesmo objecto o que possibilita a determinação da visibilidade para uma região com uma única operação. Este tipo de coerência também pode relacionar objectos que se encontram afastados na cena. Devido à projecção em perspectiva e ao ponto de vista actual, os objectos podem acabar por ser desenhados no ecrã perto uns dos outros. O algoritmo *Hierarchical Z-Buffer*, descrito no Capítulo 3, opera no espaço das imagens para calcular a visibilidade.

Se o ponto de vista sobre a cena for estático ou se mover muito lentamente, cálculos efectuados para a *frame* actual podem continuar a ser válidos para as *frames* seguintes. Esta é a ideia subjacente à coerência temporal. Em termos de determinação de visibilidade, um exemplo seria assumir que um objecto que foi identificado como ocluso na *frame* actual, provavelmente continuará ocluso nas *frames* seguintes. No entanto, a exploração da coerência temporal é complexa em grande parte porque é difícil de prever como o ponto de vista se irá mover nas *frames* seguintes ou como o conjunto de geometria visível se irá comportar, com particular relevância nas cenas dinâmicas.

2.4. SUMÁRIO

Para aumentar a eficiência das técnicas de determinação de visibilidade, a geometria da cena deve estar organizada espacialmente. As estruturas de organização de dados hierárquicas proporcionam os benefícios das estruturas em árvore. Por exemplo, a remoção de um nó implica a remoção de toda a sua árvore filha. Neste capítulo foram apresentadas algumas dessas estruturas hierárquicas, em particular as que serão referenciadas e usadas em algoritmos de remoção por oclusão descritos nos capítulos seguintes. Destaca-se também o grafo de cena que é a estrutura de organização hierárquica usada nas cenas desenvolvidas no âmbito desta dissertação.

O problema da visibilidade em cenas 3D é suficientemente complicado para não haver uma única solução algorítmica. Desta forma, a abordagem típica é a aplicação de sucessivas técnicas que resolvem a visibilidade de geometria que se enquadra em diferentes categorias. Idealmente, apenas o objecto que se encontra mais próximo do

ponto de vista deveria ser desenhado em cada pixel da imagem final. Esta diminuição da complexidade em profundidade é uma das qualidades desejáveis num algoritmo que determina a visibilidade. Para aumentar a eficiência, os algoritmos de visibilidade procuram explorar as vantagens da aplicação de diversos tipos de coerência. Desta forma, um único cálculo para determinar a visibilidade é usado em vários locais da cena amortizando o seu custo ao ser aplicado a vários objectos (coerência no espaço dos objectos) ou píxeis (coerência no espaço das imagens). O custo de cálculos de visibilidade também pode ser amortizado com o decorrer do tempo através do uso da coerência temporal.

3. REMOÇÃO POR OCLUSÃO

A remoção por oclusão pode ser descrita como um problema global a toda a cena devido à interacção espacial que pode ocorrer entre vários objectos da cena, ao contrário do que acontece com o *view-frustum culling* e a remoção de faces escondidas. Estas duas técnicas trabalham ao nível do objecto e não necessitam de ter conhecimento da localização dos restantes objectos presentes na cena para determinarem a visibilidade. O propósito deste capítulo é o de expor e analisar algumas das mais conhecidas abordagens ao problema da determinação de visibilidade que permita a remoção por oclusão.

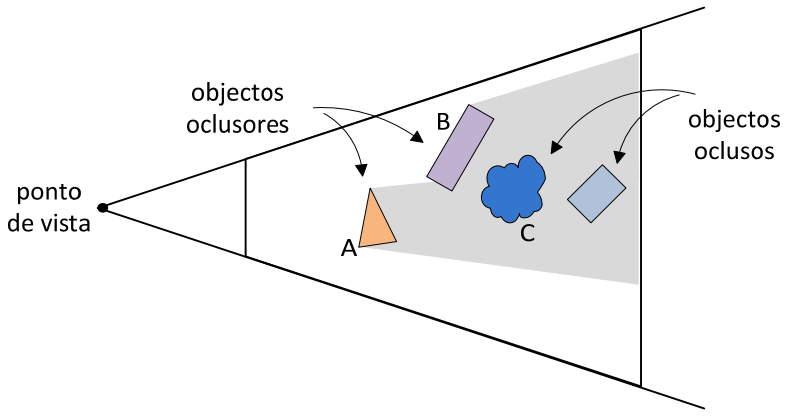


Figura 13 - Objectos oclusores e oclusos. O objecto C é ocluso devido à fusão dos oclusores A e B.

O termo oclusor representa um objecto que bloqueie da vista outro objecto no campo de visão do utilizador. Todos os objectos visíveis são potenciais oclusores. A porção da cena oclusa por um oclusor fornece uma medida do respectivo poder de oclusão. Um objecto bloqueado da vista do utilizador por um ou mais objectos é um objecto ocluso. A Figura 13 apresenta objectos que se enquadram nas duas situações descritas.

Num algoritmo de remoção por oclusão conservador é aceitável usar uma representação mais simples de um objecto (*e.g.*, o seu volume envolvente) durante o processo de cálculo de visibilidade. Ao usar o volume envolvente para verificar se o objecto se encontra ocluso poder-se-á estar a considerar um objecto visível quando na realidade se encontra ocluso. Esta situação pode ocorrer quando apenas um dos cantos do volume envolvente se encontra visível. Neste caso, a imagem final gerada é correcta apesar de o objecto ser enviado para o *hardware* gráfico desnecessariamente. Este caso de excesso de conservação é compensado pelo facto de ser usualmente mais fácil testar a visibilidade de uma forma simples como uma caixa, do que testar o objecto detalhado nela contido. A situação inversa, usar como oclusor o volume envolvente e não o objecto nele contido, pode dar origem a que um objecto seja considerado ocluso erradamente. Este problema pode ser evitado com o uso da geometria contida no volume envolvente como oclusor. Neste caso a correcção da imagem final apresentada ao utilizador é prejudicada. A Figura 14 ilustra estas duas situações que decorrem de usar volumes envolventes como oclusores e como forma de determinar se um objecto é ocluso.

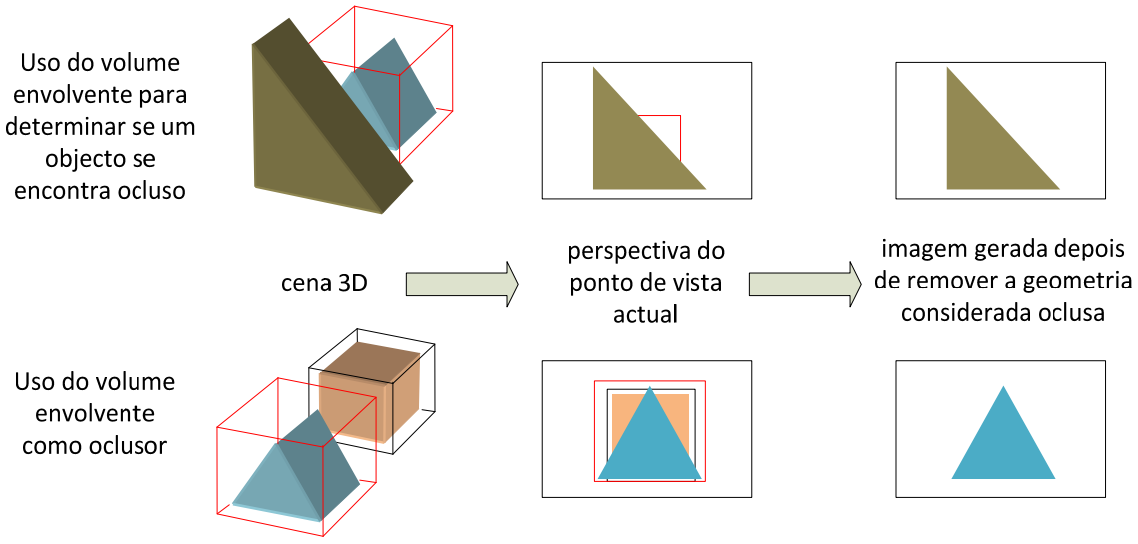


Figura 14 - Uso de volumes envolventes na determinação de visibilidade de um objecto.

Um conceito associado ao carácter global da remoção por oclusão é a fusão de oclusores. A combinação de vários objectos presentes numa cena pode oferecer um poder de oclusão superior caso fossem considerados como oclusores individuais. Um exemplo elucidativo é uma floresta. Uma árvore isolada não oferece grande poder de oclusão quando considerada isoladamente, mas a soma de muitas árvores irá ocludir bastante. Um bom algoritmo de remoção por oclusão deverá realizar a fusão de oclusores.

3.1. VISIBILIDADE EXACTA

O presente capítulo e a dissertação em geral debruça-se sobre algoritmos que permitem na fase da aplicação (CPU) eliminar objectos oclusos evitando assim o seu envio para o GPU. No entanto, existe um mecanismo presente na generalidade do actual *hardware* gráfico que garante que a imagem final é correcta. Isto é conseguido desde que todas as primitivas que contribuem para a imagem sejam enviadas para o *hardware* gráfico. Este mecanismo é denominado de *Z-Buffer* [CAT74].

O *Z-Buffer* é do mesmo tamanho que o *frame buffer*, mas em vez de conter informação sobre a cor de cada píxel guarda valores de profundidade. Estes valores de *Z* descrevem a distância a que se encontra o polígono que ocupa determinado píxel do ponto de vista actual. Quando um novo polígono é enviado para o *hardware* gráfico, a sua profundidade é testada em cada píxel contra o conteúdo actual do *Z-Buffer*. Se o valor de *Z* indicar que se encontra mais próximo do ponto de vista que o actual conteúdo (ou a região testada se encontrar vazia) o polígono é desenhado no *frame buffer*. Este processo encontra-se ilustrado na Figura 15 de forma simplificada.

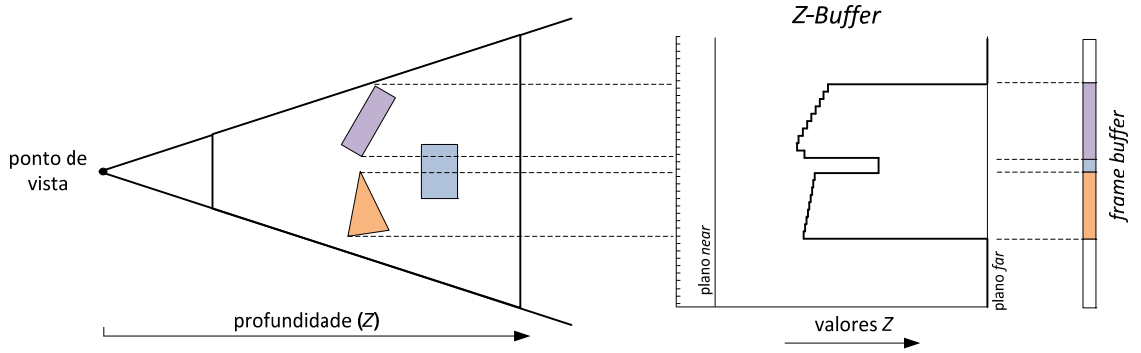


Figura 15 - Conteúdo do *Z-Buffer* com valores das profundidades de uma cena.

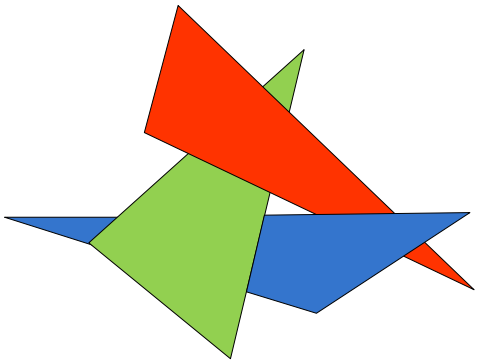


Figura 16 – Sobreposição mútua.

A visibilidade calculada ao nível do píxel permite resolver a oclusão parcial de polígonos ao evitar desenhar píxeis desnecessários, situação ilustrada na Figura 16. A sobreposição mútua é resolvida pelo *Z-Buffer* ao detectar as porções dos polígonos oclusas e assim apenas desenhar o polígono mais próximo do ponto de vista em cada píxel. Um algoritmo de remoção por oclusão que opera ao nível dos objectos não se adequa a este tipo de oclusão pois nesta situação nenhum polígono se encontra completamente ocluso.

Uma das desvantagens da técnica do *Z-Buffer* é a de usar como informação para determinar a visibilidade, primitivas que se encontram praticamente processadas e prontas para serem desenhadas. Em cenas com elevada complexidade em profundidade, muitos píxeis serão processados e a sua profundidade comparada sem que no final contribuam para a imagem final. Outro aspecto advém da ausência de retorno para a aplicação do resultado do teste de profundidade. Isto significa que todas as primitivas têm que ser enviadas para poderem ser testadas. Este problema é resolvido no actual *hardware* gráfico que permite uma malha de retorno para a aplicação através do mecanismo das *Hardware Occlusion Queries*, abordado na Secção 3.6. Este mecanismo permite testar a profundidade de geometria simples para decidir o envio de geometria complexa para a *pipeline* gráfica.

3.2. CLASSIFICAÇÃO DE ALGORITMOS DE REMOÇÃO POR OCLUSÃO

Ao longo dos anos têm sido apresentados variados algoritmos direccionados ao problema que persiste desde os inícios da computação gráfica, a determinação de visibilidade. A publicação dos detalhados estudos por Coehen-Or *et al.* [COH03] e Bittner e Wonka

[BIT03] forneceu uma taxinomia que permite a comparação das variadas abordagens ao problema da remoção por oclusão.

Ponto vs Região

Considerando o domínio da determinação de visibilidade, os diferentes métodos podem ser classificados como algoritmos que determinam a visibilidade a partir de um ponto ou de uma região, Figura 17. A maioria dos algoritmos que determinam a visibilidade para uma região trabalha numa fase de pré-processamento para determinar o conjunto potencialmente visível de objectos. Este conjunto é válido para qualquer ponto de vista no interior dessa região. Uma vantagem da visibilidade a partir de uma região é que os cálculos são válidos para um conjunto de *frames*, sendo o custo desses mesmos cálculos amortizado ao longo dessas *frames*. No entanto, os algoritmos de região usualmente requerem um longo pré-processamento e não conseguem lidar com objectos móveis na cena tão bem como os algoritmos baseados no ponto de vista.

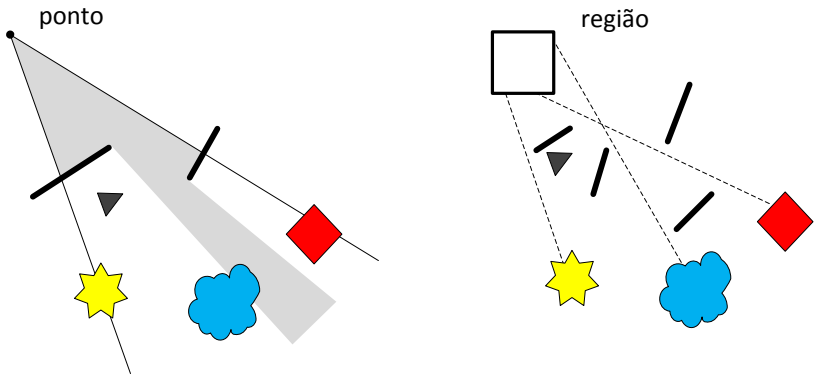


Figura 17 - Visibilidade a partir de um ponto (esquerda) e de uma região (direita).

Um exemplo da abordagem por regiões é o método de células e portais para modelos arquitectónicos [TEL91], neste caso as cenas encontram-se naturalmente organizadas em células (divisões) e interligadas por portais (portas, janelas). O algoritmo de visibilidade pré determina se outras células são visíveis apenas através dos portais. Algumas considerações podem ser feitas relativamente a esta abordagem:

- O conjunto potencialmente visível encontrado é apenas válido para a configuração original da cena para o qual foi calculado, logo pouco adequado a cenas dinâmicas;

- O conjunto potencialmente visível calculado, considerando um ponto de vista no interior de uma célula, é demasiadamente conservador. Situação ilustrada na Figura 18;

- Fazer a sua determinação para todas as células representa uma tarefa computacionalmente dispendiosa, realizada em pré-processamento significa longos períodos de inicialização da aplicação.

- É difícil implementar a determinação de um conjunto potencialmente visível preciso para uma cena de carácter geral. Enquanto em modelos de interiores de edifícios as paredes actuam como divisores naturais, em cenas exteriores a tarefa pode revelar-se mais difícil.

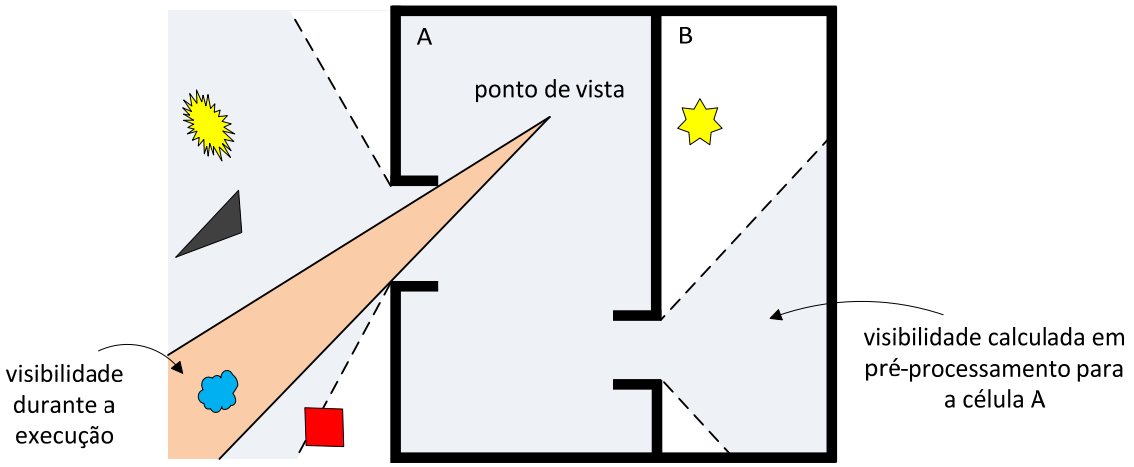


Figura 18 – Visibilidade em *runtime* comparada com a visibilidade pré calculada de uma célula.

Os métodos que determinam a visibilidade com base num ponto necessitam de recalcular a informação de visibilidade sempre que o ponto de vista se move, na realidade, estes algoritmos determinam a visibilidade todas as *frames*. Exemplos deste tipo de algoritmos são [GRE93, ZHA97, WON00, KLO01, HIL02, BIT04, GUT06, MAT08]. Apesar de necessitarem de mais cálculos em cada *frame* funcionam em *runtime* e apresentam uma maior flexibilidade o que permite cenas dinâmicas. Estes algoritmos realizam a fusão de oclusores devido a operarem essencialmente no espaço das imagens. Também resolvem os quatro pontos anteriormente enumerados relativos à determinação do conjunto potencialmente visível de geometria a partir de uma região.

Objecto vs Imagem

Os métodos de remoção por oclusão que trabalham no espaço dos objectos realizam os cálculos necessários com as coordenadas 3D originais dos objectos para obterem o conjunto de polígonos visíveis ou oclusos. Os métodos que usam o espaço das imagens, por outro lado, operam na representação discreta (píxeis) dos objectos resultante do processo de rasterização (conversão da informação vectorial dos objectos em píxeis realizada pelo *hardware* gráfico). Estes métodos usam os valores projectados das coordenadas 3D originais dos objectos. A vantagem do espaço das imagens resulta das operações serem realizadas sobre um conjunto discreto de resolução finita, também tendem a ser mais simples e robustos quando comparados com os métodos que usam o espaço dos objectos [COH03].

3.3. ALGORITMO DE REMOÇÃO POR OCLUSÃO GENÉRICO

No âmbito desta dissertação apenas se consideram algoritmos que determinam a visibilidade a partir de um ponto, durante o cálculo de uma *frame*. A Figura 19 apresenta um pseudo-código para este tipo de algoritmo de remoção por oclusão onde a função `estaOcluso()`, geralmente chamada de teste de visibilidade ou teste de oclusão, verifica se um objecto se encontra ocluso. A variável *G* representa o conjunto de objectos a desenhar durante uma *frame*.

```
AlgoritmoRemocaoOclusao(G)
para cada objecto g ∈ G
{
    se ( não estaOcluso(g,G))
        desenha(g)
}
```

Figura 19 - Pseudo-código para um algoritmo de Remoção por Oclusão genérico (1ª versão).

Em cada *frame*, depois de aplicadas outras técnicas de visibilidade já discutidas, resulta um conjunto de objectos que deverão ser desenhados, *G*. Cada objecto (*g*) desse grupo é testado relativamente a todos os objectos em *G*, potenciais oclusores. Se o teste determinar que o objecto *g* não se encontra ocluso a função `desenha()` é executada.

O algoritmo genérico apresentado apenas considera o poder oclusor de cada objecto individualmente para além de que o número de testes aumenta de forma quadrática com o número de objectos em *G*. Uma forma de contornar esta limitação, e diminuir a quantidade de testes necessários, será manter uma representação de oclusão (O_R). Neste caso, O_R constitui o conjunto de objectos oclusores (depois de projectados no espaço das imagens) e é actualizado como cada novo objecto ($g \in G$) que é considerado visível. Desta forma, todos os objectos que são considerados visíveis são usados como potenciais oclusores de próximos objectos a testar. O algoritmo apresentado na Figura 20 demonstra este funcionamento.

```
AlgoritmoRemocaoOclusao(G)
O_R = vazio
para cada objecto g ∈ G
{
    se ( não estaOcluso(g,O_R))
    {
        desenha(g)
        adiciona(g,O_R)
    }
}
```

Figura 20 - Pseudo-código para um algoritmo de Remoção por Oclusão genérico (2ª versão).

A ordenação da frente para trás dos objectos *g*, relativamente ao ponto de vista, permite que a função `estaOcluso()` determine a visibilidade por ordem crescente de probabilidade de se encontrarem oclusos. Se um objecto *g* for determinado como ocluso é removido de posteriores processamentos. A partir deste momento sabe-se que não contribuirá para a imagem final nem para ocludir outros objectos. Caso se determine que o objecto se encontra visível este é desenhado, contribuindo provavelmente para a imagem final. Um objecto visível é adicionado a O_R para que o seu poder oclusor seja considerado nos testes seguintes.

Um factor importante na performance da maioria dos algoritmos de remoção por oclusão é a ordem pela qual os objectos são testados. Como exemplo, considere o modelo 3D de um automóvel com motor. Se o exterior do automóvel for desenhado em primeiro lugar, então o motor (provavelmente) será determinado como ocluso. Por outro lado, se o

motor for desenhado primeiro, não contribuirá para a imagem final mas será enviado para o *hardware* gráfico, visto que não foi possível identifica-lo como ocluso. Assim, o desempenho pode ser melhorado com a ordenação de frente para trás dos objectos a desenhar relativamente à sua distância ao ponto de vista actual sobre a cena. Com esta ordenação cada objecto será sempre testado contra outros objectos potenciais oclusores.

3.4. *HIERARCHICAL Z-BUFFER*

O algoritmo *Hierarchical Z-Buffer* (HZB) introduzido por Greene *et al.* [GRE93] em 1993 é um algoritmo bastante importante que influenciou a investigação no âmbito da remoção por oclusão, apesar de nunca ter sido implementado na sua totalidade em *hardware*. Este algoritmo usa duas estruturas hierárquicas, uma *octree* no espaço dos objectos para organizar a cena e uma *Z-Pyramid* no espaço das imagens para guardar a informação de profundidade.

A *Z-Pyramid* é uma pirâmide de imagens onde a base é o normal *Z-Buffer* que contém as distâncias à câmara, profundidade, dos polígonos desenhados no ecrã. A base da pirâmide é o nível de maior resolução. Os restantes níveis contêm o valor de *Z* mais afastado na correspondente janela 2x2 do nível adjacente de maior resolução. Assim, cada valor de *Z* representa a geometria mais afastada para uma região quadrada do ecrã. Para manter a *Z-Pyramid* actualizada, sempre que o *Z-Buffer* é reescrito as alterações são propagadas pelos níveis de menor detalhe da *Z-Pyramid* até que seja alcançada a imagem no topo da pirâmide, onde existe apenas um valor de *Z*. O processo de formação de uma *Z-Pyramid* é ilustrado na Figura 21.

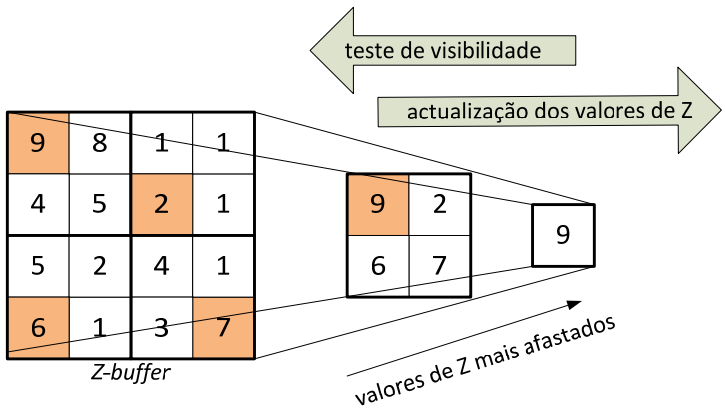


Figura 21 - *Z-Buffer* hierárquico ou *Z-Pyramid*.

Numa etapa de pré-processamento a cena é organizada numa *octree*. Durante a execução os nós da *octree* a desenhar são ordenados da frente para trás. O teste de visibilidade é realizado testando o volume envolvente de um nó da *octree* relativamente à *Z-Pyramid*. O teste inicia-se no nível da *Z-Pyramid* de resolução mais grosseira que inclua a projecção no ecrã do volume envolvente do nó em teste. Para determinar a visibilidade é usado o valor de *Z* mais próximo do volume envolvente e comparado com a *Z-Pyramid*, começando pelo topo. Se o valor no topo da *Z-Pyramid* se encontrar mais próximo que o valor de *Z* da nó em teste, então este encontra-se ocluso. Este teste continua recursivamente ao longo da pirâmide até que se saiba que o volume envolvente do nó está ocluso, ou até se atingir o nível de maior resolução (base) da pirâmide, onde se conclui que é visível. Para nós da *octree* com volumes envolventes visíveis, o teste continua recursivamente para os seus filhos. No final a geometria potencialmente visível é desenhada na *Z-Pyramid* para actualizar os valores de *Z* o que permite que posteriores testes usem o poder de oclusão da geometria previamente desenhada.

Neste algoritmo a coerência temporal é explorada através do uso de uma lista que consiste nos nós da *octree* determinados como visíveis na *frame* anterior. Os nós presentes nesta lista são inicialmente desenhados sem que lhes seja aplicado o teste de visibilidade, fornecendo assim uma fonte inicial de dados para a construção da *Z-Pyramid*. Em seguida o algoritmo percorre a *octree* evitando os nós já desenhados. Este segundo passo permite preencher as partes em falta da cena, adicionado nós visíveis à lista de coerência temporal. O passo final é a actualização da lista onde todos os nós presentes são testados contra a *Z-Pyramid* para que seja possível remover nós que na *frame* actual se tenham tornado oclusos.

Uma versão deste algoritmo que usa uma pirâmide com alguns níveis foi implementada em *hardware*, entre outros, pela ATI com a denominação de HyperZ [PAB02]. A base desta tecnologia é uma *Z-Pyramid* de três níveis com regiões de 8x8 píxeis.

Conceptualmente o algoritmo HZB apresenta bastante potencial ao aplicar-se a cenas genéricas e permitir a fusão de oclusores, a exploração da coerência temporal, espacial e de imagem. No entanto, actualmente não é totalmente implementado em *hardware* o que torna, em grande parte, impraticável o seu uso em tempo real.

3.5. *HIERARCHICAL OCCLUSION MAP*

Outro algoritmo que opera no espaço da imagem é o *Hierarchical Occlusion Map* (HOM) apresentado por Zhang *et al.* [ZHA97]. A cena é particionada usando uma Hierarquia de Volumes Envolventes em pré-processamento, onde os principais oclusores da cena são também detectados e inseridos numa base de dados de oclusores. Esta pré-selecção de oclusores indicia que este algoritmo não é adequado para cenas dinâmicas. Depois de encontrado este conjunto de oclusores, estes são desenhados sem que lhes seja realizado qualquer teste de oclusão. Em seguida é processada a restante cena onde é realizado um teste para determinar a visibilidade de cada objecto. O algoritmo divide o teste de visibilidade em duas partes: um teste de profundidade unidimensional na direcção *Z* e um teste de sobreposição bidimensional no plano *XY*, *i.e.*, no espaço da imagem.

Para o teste de sobreposição bidimensional, em cada *frame*, um subconjunto de oclusores da base de dados é escolhido usando heurísticas tal como a posição do observador e as dimensões do *view-frustum*. Estes oclusores são desenhados no *frame buffer* como desenhos a preto e branco. Este processo fornece as silhuetas dos oclusores e as regiões do *frame buffer* que eles ocupam. A imagem obtida é chamada de mapa de oclusão. Tal como no algoritmo HZB, uma vantagem desta operação é que vários pequenos oclusores podem ser combinados num oclusor maior. Nesta fase é lido o *frame buffer* e construída uma pirâmide de imagens onde cada nível é constituído pelo valor resultante da média de quadrados de 2x2 píxeis do nível de resolução imediatamente superior. A pirâmide resultante é denominada de mapa de oclusão hierárquico. O processo é semelhante ao do algoritmo HZB, neste caso a pirâmide é construída com base na média dos quadrados e não no valor máximo. Esta operação pode ser realizada em *hardware* que suporte *mipmapping*[1], uma funcionalidade comum no *hardware* gráfico corrente.

[1] Geração de um conjunto de imagens (pré calculadas) de menor detalhe que acompanha uma textura principal para, entre outros aspectos, melhorar a velocidade de *rendering*.

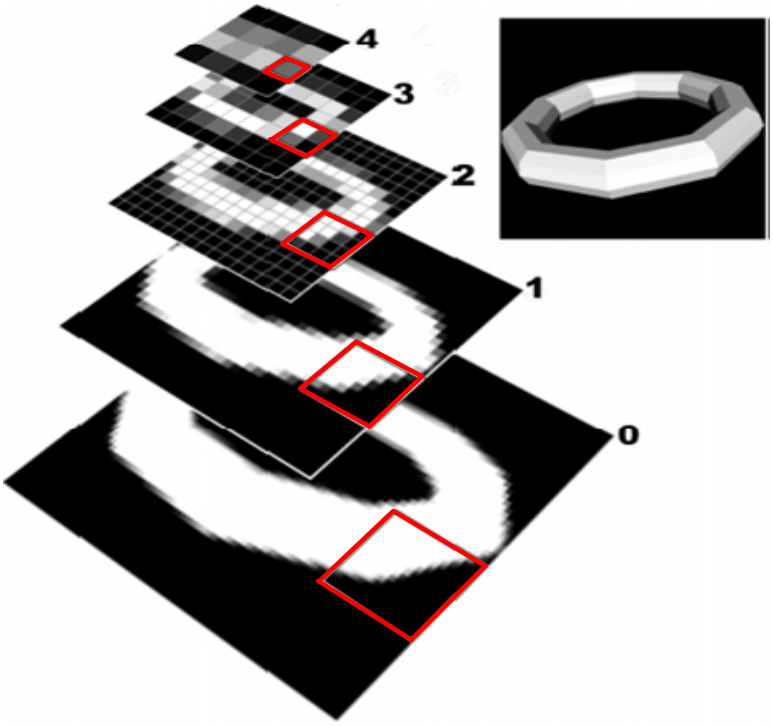


Figura 22 - Mapa de oclusão hierárquico. Ilustração obtida de [ZHA97].

A Figura 22 apresenta um exemplo de um mapa de oclusão hierárquico. O quadrado saliente denota a relação entre os píxeis do nível de maior resolução (nível 0) e os píxeis dos restantes níveis.

O teste de sobreposição no espaço da imagem é realizado de forma semelhante à do teste do algoritmo HZB descrito na secção anterior. Se todos os píxeis usados na comparação da silhueta da geometria testada com o mapa de oclusão hierárquico forem opacos (brancos), significa que a projecção do volume envolvente da geometria coincide com a dos oclusores. Por outro lado, se um píxel não for opaco, então o teste para esse píxel continua recursivamente até que se atinja a base da pirâmide.

Quando o resultado do teste de sobreposição retorna opaco é necessário aplicar o teste de profundidade para determinar se a geometria se encontra atrás ou à frente dos seus potenciais oclusores. Para manter a informação de profundidade necessária para este teste uma segunda estrutura de dados é usada, o *depth estimation buffer*. Esta estrutura de dados em *software* guarda os valores de profundidade mais afastados dos oclusores e tem normalmente uma resolução inferior à do *frame buffer.* O teste de profundidade consiste na comparação do conteúdo do *depth estimation buffer*, coincidente com a

projecção do volume envolvente do objecto testado, com o valor de profundidade mais afastado do volume envolvente do objecto. Se o teste indicar que a caixa envolvente está localizada à frente dos oclusores, o teste falha.

Um objecto é considerado ocluso se os testes de sobreposição e profundidade passarem. Se qualquer um dos testes falhar, o objecto é desenhado. No entanto, em nenhuma das situações o mapa de oclusão hierárquico é actualizado. Este funcionamento não incremental é uma clara desvantagem deste algoritmo. Outro aspecto menos positivo é a dependência da escolha de bons oclusores em cada *frame*. Este não é um problema trivial especialmente em situações onde o pré-processamento deverá ser evitado para permitir cenas dinâmicas. A coerência temporal não é explorada e o processo de leitura do *frame buffer* para a realização do teste de sobreposição introduz uma latência significativa. Como vantagem este algoritmo realiza a fusão de oclusores.

Mais recentemente, Aila e Miettinen [AIL00, AIL04] apresentaram um algoritmo derivado do HOM, denominado de *Incremental Occlusion Map* (IOM). Como descrito anteriormente, uma das desvantagens do algoritmo HOM é o facto de usar o *frame buffer* para criar o mapa de oclusão e a necessidade de o ler de volta para conseguir realizar os teste de visibilidade. No algoritmo IOM, este passo é evitado realizando a extracção das silhuetas dos oclusores no espaço dos objectos através de uma operação em *software*. Este algoritmo em conjunto com outras técnicas demonstra um bom desempenho quando usado em *runtime*, no entanto, é bastante complexo de implementar.

3.6. TESTE DE OCLUSÃO POR *HARDWARE* BASEADO EM OPENGL

Como visto até agora, a questão central de um algoritmo de remoção por oclusão é saber se um determinado objecto irá contribuir para a imagem final caso seja desenhado. Actualmente, esta questão pode ser colocada directamente ao *hardware* através de um mecanismo chamado genericamente de *Hardware Occlusion Query*, doravante denominado por **teste de oclusão por *hardware***. A ideia de acelerar o processo de determinação de visibilidade recorrendo a *hardware* dedicado em algoritmos que operam no espaço das imagens foi sugerida por Greene *et al.* [GRE93]. Mais tarde Bartz *et al.* [BAR98] propôs uma implementação detalhada deste mecanismo.

De uma forma simples, com os testes de oclusão por *hardware*, a aplicação pode questionar o *hardware* gráfico para saber se um determinado conjunto de polígonos é visível quando comparados com o conteúdo actual do *Z-Buffer*. Esses polígonos são frequentemente os volumes envolventes de um objecto mais complexo. Se nenhum desses polígonos for visível, então o objecto pode ser removido de posteriores processamentos. A implementação em *hardware* realiza a rasterização dos polígonos e se durante este processo for despoletada uma escrita no *Z-Buffer* significa que pelo menos um píxel é visível. O resultado do teste de oclusão por *hardware* pode ser obtido pela aplicação através de uma extensão de OpenGL dedicada. A Figura 23 ilustra o princípio de funcionamento e uso do teste de oclusão por *hardware* para decidir a visibilidade de um objecto durante a execução da aplicação. A numeração indica a sequência das etapas.

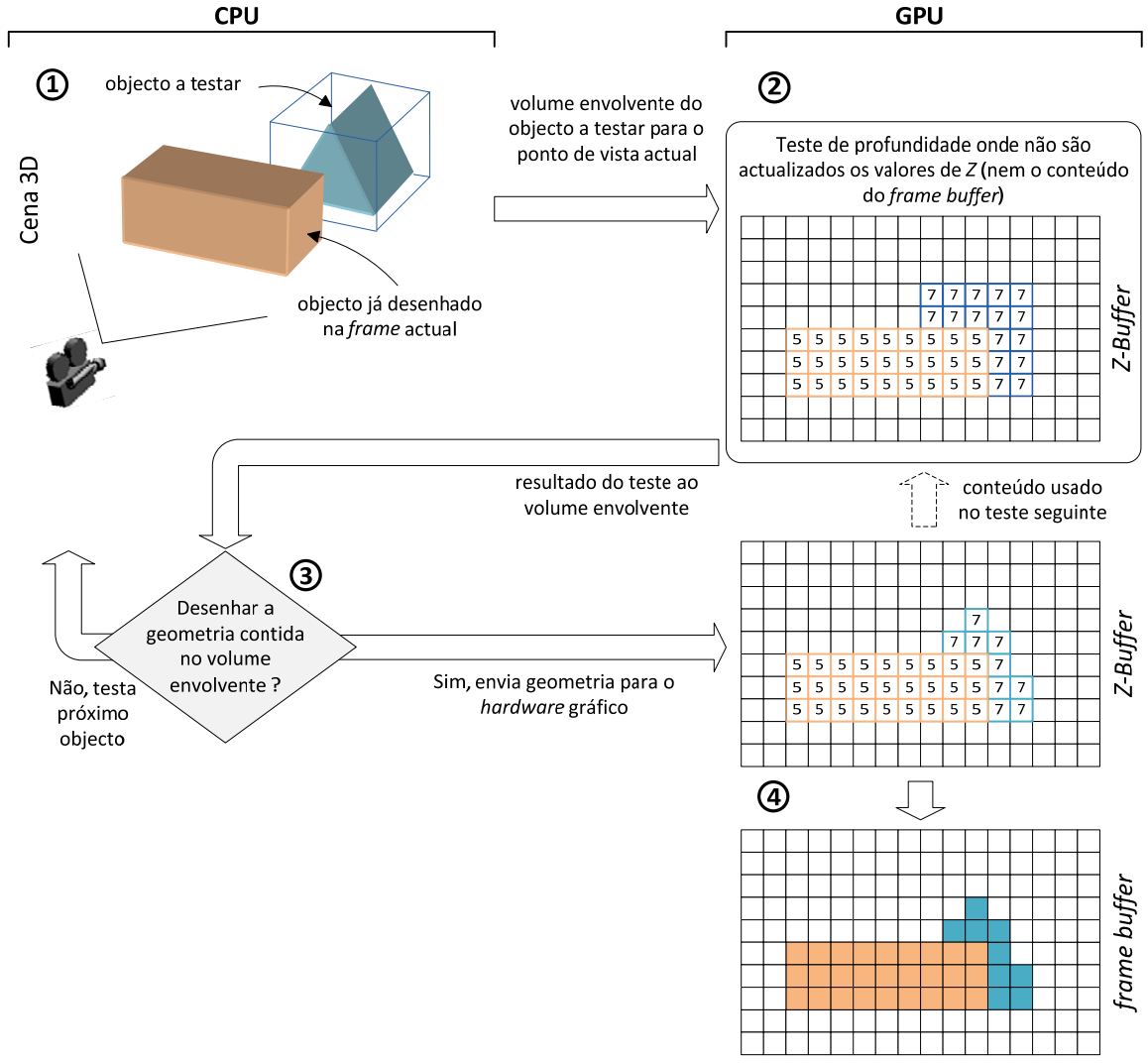


Figura 23 - Teste de oclusão por *hardware* para determinar a visibilidade de um objecto.

Possivelmente o primeiro *hardware* gráfico corrente com este tipo de mecanismo foi o VISUALIZE fx da Hewlett-Packard [SCO98] que disponibilizava a extensão de OpenGL *GL_HP_occlusion_test* [HP97]. Mais tarde, em 2002, foi introduzida pela NVIDIA com a placa aceleradora gráfica GeForce3 uma extensão mais evoluída denominada de *GL_NV_occlusion_query* [NV01]. Actualmente também existe como uma extensão oficial ARB [ARB03].

O teste de oclusão por *hardware* introduzido pela Hewlett-Packard apresenta duas grandes limitações [AKE08]. Em primeiro lugar apenas retorna verdadeiro ou falso para indicar a visibilidade do objecto questionado. Este resultado binário impede que um objecto que apenas contribua com alguns píxeis para a imagem final possa ser removido na sua totalidade se for considerado que a imagem final não é significativamente prejudicada. Em segundo lugar, a extensão *GL_HP_occlusion_test* é implementada como um mecanismo de pára-e-espera, a aplicação deverá suspender a sua execução enquanto espera pelo resultado do teste. Esta característica impede que os testes sejam serializados o que condiciona a exploração do paralelismo entre CPU e GPU.

A introdução da extensão pela NVIDIA veio resolver as limitações da extensão da Hewlett-Packard. Com a *GL_NV_occlusion_query* é possível realizar testes em série e obter os respectivos resultados numa fase posterior. Assim o CPU fica livre para realizar outros processamentos necessários à aplicação. A *GL_NV_occlusion_query* também retorna um número inteiro que indica quantos píxeis passaram no teste de profundidade, *i.e.*, a quantidade de píxeis visíveis. Se esse número for inferior a um determinado limite de píxeis, então algumas aplicações podem decidir não realizar o *rendering* dos objectos desse volume envolvente. Neste caso sacrifica-se a correcção da imagem final pelo desempenho. Quando os testes de oclusão por *hardware* são abordados no contexto da computação gráfica moderna, a extensão da NVIDIA está usualmente implícita.

Apesar da evolução dos testes, o seu uso em algoritmos de remoção por oclusão continua a não ser trivial devido ao custo que representam para a aplicação. Para além do custo associado ao teste em si, existe a latência entre a realização do teste e a disponibilidade do resultado. Esta latência decorre do atraso no processamento do teste em longas *pipelines* de *rendering*, o tempo de processamento do teste e o custo de devolver o

resultado ao CPU. Esta situação causa dois problemas quando os testes são usados de forma sequencial [BIT04]: CPU *stalls* e GPU *starvation*.

Após a emissão de um teste o CPU fica à espera do seu resultado e não fornece mais dados ao GPU para processamento. Quando o resultado fica finalmente disponível, a *pipeline* do GPU poderá já estar vazia. Como consequência, o GPU terá que esperar que o CPU processe o resultado do teste antes que este lhe envie mais trabalho. Para evitar os CPU *stalls*, a melhor estratégia é preencher os tempos de latência com outros processamentos necessários à aplicação, incluindo a realização de novos testes de oclusão por *hardware*. Como estes podem ser realizados simultaneamente, não é necessário esperar pelo resultado antes de realizar novo teste, sendo o seu resultado usado posteriormente quando disponível [KOV05].

A Figura 24 ilustra dois esquemas diferentes de gestão de testes de oclusão por *hardware*. As tarefas Dn, Tn e Rn representam o *rendering*, teste de oclusão por *hardware* e remoção por oclusão para cada nó n processado. O *esquema 2* apresenta um possível escalonamento de tarefas para preencher os tempos de latência usando os resultados dos testes quando disponíveis. Neste esquema o nó 2 é assumido como visível, procedendo com o seu render antes mesmo da disponibilidade do resultado do teste.

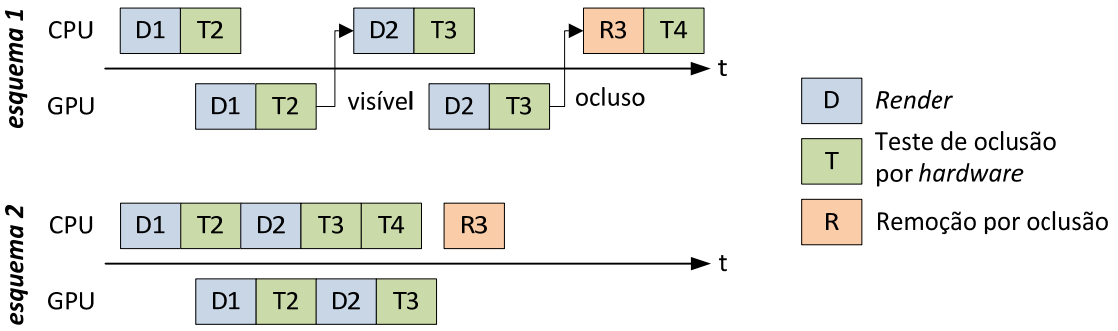


Figura 24 - Esquemas de gestão de testes de oclusão por *hardware*.

Apesar dos problemas descritos, o uso dos testes de oclusão por *hardware* como mecanismo que permite tomar a decisão de remover geometria oclusa durante a execução de uma aplicação apresenta várias vantagens:

- Qualquer objecto pode ser usado como oclusor. Para o teste é usado o conteúdo do *Z-Buffer*, desta forma qualquer geometria da cena que já tenha sido desenhada na altura do teste funciona como oclusor;

- Fusão de oclusores implícita, pelo princípio descrito no ponto anterior;

- A flexibilidade de oclusores também se aplica à geometria oclusa, visto que para o teste podem ser usados os volumes envolventes ou a geometria em si.

- Como é um mecanismo simples, pode ser facilmente integrado em algoritmos de *rendering* existentes. O custo de implementações baseadas neste mecanismo é bastante inferior a algoritmos que recorrem apenas ao CPU para determinação de visibilidade.

A simplicidade dos testes de oclusão por *hardware* tornou a determinação de visibilidade em tempo real bastante tentadora. No entanto, as operações envolvidas (rasterização e retorno do resultado à aplicação) apresentam um custo que não deverá ser descurado. Os algoritmos que usam testes de oclusão por *hardware* tentam minimizar o seu uso recorrendo a várias técnicas, incluindo a coerência temporal. O desempenho também pode ser melhorado através do uso de volumes envolventes mais justos, como descrito em [BAR05].

A ordem de processamento dos objectos da frente para trás também é crítica para o desempenho dos algoritmos que usam testes de oclusão por *hardware*. No entanto, nas aplicações gráficas actuais existem outros factores a ter em consideração no momento de ordenar os objectos a processar, destes destacam-se as alterações de estado e em particular as mudanças de *shader*[2]. Enviar para a *pipeline* gráfica objectos com o mesmo estado é benéfico para o desempenho, no entanto este agrupamento poderá entrar em conflito com a ordenação da frente para trás desejável em algoritmos que usam os testes de oclusão por *hardware*.

[2] Conjunto de instruções usadas para calcular efeitos de *rendering* (*e.g.*, cor, iluminação, textura) em GPUs equipados com *pipeline* de *rendering* programável. Existem várias linguagens específicas para *shaders*, uma dela é a GLSL (http://www.lighthouse3d.com/opengl/glsl/).

3.7. SUMÁRIO

No presente capítulo foi apresentada a visibilidade exacta que permite resolver o problema da oclusão numa fase e a um nível em que os algoritmos de remoção por oclusão não operam. Neste sentido o *Z-Buffer* é, em última instância, o mecanismo que permite que a imagem apresentada ao utilizador seja correcta desde que todas as primitivas que contribuem para a imagem sejam enviadas para o *hardware* gráfico. Por outro lado, os algoritmos de remoção por oclusão procuram reduzir a quantidade de trabalho a que o *hardware* gráfico é sujeito através da identificação e remoção de geometria oclusa na fase da aplicação.

Dos três métodos que permitem remover geometria oclusa apresentados neste capítulo, o teste de oclusão por *hardware* baseado em OpenGL é o que apresenta maior potencial para realizar a remoção deste tipo de geometria. No entanto, a sua aplicação deverá ser feita de forma a evitar quebras na *pipeline* e ter em consideração os efeitos da latência associada ao seu uso. No capítulo seguinte serão apresentados vários algoritmos que usam os testes de oclusão por *hardware* para assistir a decisão de remover geometria por oclusão.

4. USO DE TESTES DE OCLUSÃO POR *HARDWARE*

O presente capítulo destina-se a descrever algoritmos que fazem uso do mecanismo de teste de oclusão por *hardware* para obterem informação relativamente ao estado de visibilidade dos objectos a desenhar. Comum a todos os algoritmos, é a preocupação em minimizar os efeitos introduzidos pela latência que existe entre a realização do teste e a obtenção do seu resultado. Os vários algoritmos são apresentados por ordem cronológica.

4.1. *CONSERVATIVE PRIORITIZED-LAYERED PROJECTION*

O algoritmo *Conservative Prioritized-Layered Projection* (cPLP) apresentado por Klosowski e Silva [KLO01] que usa testes de oclusão por *hardware* é uma versão conservadora do algoritmo *Prioritized-Layered Projection* (PLP). O algoritmo PLP [KLO00], dos mesmos autores, é apresentado em primeiro lugar.

O algoritmo PLP é baseado em *software* e tenta fazer o *render* da cena utilizando apenas uma quantidade pré-definida de polígonos. Apesar de organizar a cena numa *octree*

outras estruturas de dados podem ser usadas. No entanto, é necessário ter em consideração que no interior do *view-frustum* o algoritmo não funciona de forma hierárquica, apenas trabalha com os nós folha que contêm a geometria. Numa fase de pré-processamento a *octree* é percorrida para atribuir prioridades aos nós. A heurística usada para o efeito baseia-se no conceito de solidez, significando que nós que contenham muita geometria são considerados mais sólidos.

Durante a execução do ciclo de *rendering*, o algoritmo usa a solidez em combinação com o ponto e direcção de visualização actual para gerar uma lista, ordenada por prioridades, de nós provavelmente visíveis. A prioridade de um nó, deste modo reflecte a solidez acumulada dos nós que se encontram entre ele próprio e o ponto de vista. Se uma elevada solidez foi acumulada, é provável que esse nó se encontre ocluso e consequentemente é lhe dada uma baixa prioridade. Este algoritmo não é conservador pois o *rendering* dos nós presentes na lista termina quando um limite definido de polígonos é atingido.

Outra heurística para definir a prioridade de um nó seria atribuir a prioridade de -1 ao nó que contém o ponto de vista, -2 aos seus vizinhos, -3 aos vizinhos destes e assim sucessivamente. Isto fará com que as prioridades atribuídas aos nós se propaguem de uma forma regular à medida que se afastam do ponto de vista do utilizador. Esta heurística é fácil de implementar e proporciona resultados relativamente bons [COR02].

Quando o algoritmo PLP termina o *render* de uma *frame* por ter atingido o limite de polígonos, o *Z-Buffer* contém os valores de Z da geometria desenhada e a fila contém os nós que o algoritmo teria processado caso não houvesse limite. Como descrito em [KLO01], a versão conservadora do algoritmo PLP é conseguida através da aplicação de testes de oclusão por *hardware* aos nós ainda presentes na fila o que permite desenhar os restantes nós visíveis. A principal limitação deste algoritmo reside na etapa de pré-processamento que limita o seu uso a cenas estáticas.

4.2. *FAST AND SIMPLE OCCLUSION CULLING*

Hillesland *et al.* [HIL02] apresentou um algoritmo denominado de *Fast and Simple Occlusion Culling* (FSOC). Este algoritmo é um dos primeiros a usar a extensão da NVIDIA

e divide a cena usando uma estrutura em grelha regular. Na altura do *render* de uma *frame* a grelha é processada por filas no sentido de frente para trás relativamente ao ponto de vista actual. Os autores salientam o problema da latência dos testes de oclusão por *hardware* e a solução por eles apresentada é de simplesmente realizar *N* testes de cada vez e depois obter as correspondentes *N* respostas, onde *N* é um número que tem que ser ajustado manualmente para se atingir a melhor performance possível.

Os resultados das simulações realizadas comparam o desempenho do FSOC com o uso apenas do *view-frustum culling* e revelam uma melhoria na performance. No entanto, a estrutura de dados em grelha não se aplica a cenas onde a densidade de geometria varia e existem paramentos que necessitam de ser ajustados manualmente para cada caso individualmente, tal como o número de testes a realizar de cada vez e a resolução da grelha.

4.3. *COHERENT HIERARCHICAL CULLING*

O algoritmo *Coherent Hierarchical Culling* (CHC) apresentado por Bittner *et al.* [BIT04] em 2004 explora a coerência espacial e temporal para minimizar os efeitos dos testes de oclusão por *hardware*. Em cada *frame*, o varrimento da estrutura de dados em árvore termina ou em nós folha ou em nós internos oclusos. Estes nós formam um conjunto denominado de nós terminadores, *T*. Os restantes nós internos classificados como visíveis são denominados de nós abertos, *O*.

Na *frame* i, o algoritmo não testa a oclusão dos nós abertos em O_{i-1}, fazendo-o apenas a todos os nós terminadores em T_{i-1}. Esta abordagem assume que um nó aberto visível na *frame* anterior tende a permanecer visível na *frame* actual. Do conjunto de nós terminadores, se um nó se encontrava visível na *frame* anterior é realizado o teste de oclusão por *hardware* e prossegue-se imediatamente para o *render* da geometria. Se um nó se encontrava ocluso, é realizado o teste de oclusão por *hardware* adiando o processamento da árvore filha para quando o resultado se encontrar disponível.

Se o estado de visibilidade de um nó se modifica na *frame* actual o conjunto de nós terminadores é actualizado através de um método de propagação de visibilidade, onde o

estado de um nó pai é alterado de acordo com o estado de visibilidade dos seus nós filhos. Desta forma não é necessário manter um registo explícito dos conjuntos O_i e T_i.

A Figura 25 ilustra a alteração de visibilidade para uma hierarquia exemplo. Na *frame 0*, à esquerda, são representados os nós terminadores aos quais será aplicado o teste de oclusão por *hardware* na *frame 1*. Nesta *frame*, à direita, todos os nós aos quais foi aplicado o teste de oclusão por *hardware* são representados por um contorno a tracejado. Com os resultados dos testes disponíveis o conjunto de nós terminadores é actualizado. Do lado esquerdo da hierarquia da *frame 1*, o estado de visibilidade do nó pai transita para ocluso através do mecanismo de propagação devido a todos os seus filhos se encontrarem oclusos. Do lado direito, como o teste de oclusão por *hardware* revelou que o nó se encontrava visível, foi processada a sua árvore filha realizando testes de oclusão por *hardware*.

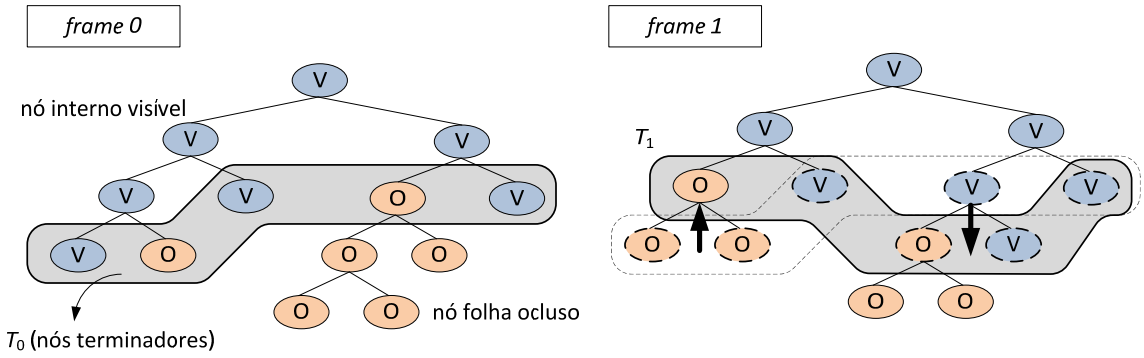


Figura 25 - Exemplo de uma hierarquia sujeita ao algoritmo CHC em duas *frames* consecutivas.

O algoritmo CHC reduz o número de testes de oclusão por *hardware* necessários ao explorar a coerência espacial. Como resultado da propagação do estado de visibilidade, todo o nó terminador ocluso tem sempre um pai visível. Isto significa que as partes oclusas da cena são representadas com um número mínimo de nós. É a estes nós terminadores que o algoritmo realiza os testes de oclusão por *hardware* e assim evita processar as correspondentes árvores filhas. Assim, o número de testes realizados é o mínimo possível. A coerência temporal é também explorada através da não realização de testes de oclusão por *hardware* aos nós abertos (anteriormente visíveis), o que implica uma redução no número de testes por um factor de $1/k$, onde k é um parâmetro relacionado com a ramificação da hierarquia [BIT04].

Para minimizar os efeitos da latência associada aos testes de oclusão por *hardware* o algoritmo CHC não fica simplesmente à espera que o *hardware* retorne o resultado. Em vez disso, os testes são intercalados com o *rendering* da geometria anteriormente visível. O comportamento intercalado é conseguido pelo uso de uma fila onde são colocados os nós depois de testados. Esta fila é monitorizada continuamente para verificar se o resultado do teste relativo ao nó que se encontra no inicio da fila já se encontra disponível. Caso contrário, o algoritmo prossegue com o *rendering* de geometria determinada como visível na *frame* anterior. Com esta abordagem, o resultado dos novos testes ficará entretanto disponível e a *pipeline* não ficará suspensa.

Algumas optimizações possíveis são sugeridas pelos autores, apesar do algoritmo base já permitir melhorias substanciais de performance quando comparado com o *view-frustum culling*. Sem necessidade de pré-processamento ou afinação para cenas específicas, o algoritmo CHC é adequado para todo o tipo de cenas e a sua implementação não é complexa. Como desvantagens, salienta-se a realização de alguns testes de oclusão por *hardware* desnecessários. Em cada *frame* é necessário realizar testes de oclusão por *hardware* a todos os nós folha anteriormente visíveis de forma a actualizar correctamente o estado de visibilidade na hierarquia através do método de propagação. Este problema foi parcialmente resolvido por Staneker *et al.* [STA04] que propôs um método que evita testes a objectos que são decididamente visíveis. Este método usa diversos testes em *software* (*e.g.*, Mapa de Ocupação) para manter um mapeamento das zonas do ecrã já desenhadas. No entanto, o custo destes testes em *software* é na maioria das vezes superior ao custo do teste de oclusão por *hardware* [GUT06].

Outra situação ocorre nos nós internos removidos por se encontrarem fora do *view-frustum* na *frame* anterior. Neste caso, para actualizar a hierarquia é necessário realizar um teste de oclusão por *hardware* a todos os nós folhas que estavam fora do *view-frustum* na *frame* anterior. Este comportamento levará certamente à realização de testes desnecessários [GUT06]. O algoritmo também apresenta limitações em cenas altamente dinâmicas e com rápidos movimentos do ponto de vista porque os nós terminadores nestas situações irão variar com bastante frequência, reduzindo a eficiência na determinação de visibilidade. No entanto, este é um problema que se aplica a qualquer algoritmo que faça uso da coerência temporal. Além deste problema, em cenas com baixa

complexidade em profundidade, o custo associado à realização de testes de oclusão por *hardware* pode degradar o desempenho de tal forma que este se torna pior do que em situações onde apenas é usado o *view-frustum culling*.

4.4. *NEAR OPTIMAL HIERARCHICAL CULLING*

Apresentado em 2006 por Guthe *et al.* [GUT06], o algoritmo *Near Optimal Hierarchical Culling* (NOHC) partilha semelhanças com o algoritmo CHC mas adopta uma abordagem mais analítica aos problemas associados ao uso de testes de oclusão por *hardware*.

Ao contrário do algoritmo CHC, os testes de oclusão por *hardware* também podem ser aplicados a nós fora do conjunto de nós terminadores. O princípio de funcionamento do algoritmo baseia-se na aplicação de uma função custo/benefício a todos os nós que se encontram no *view-frustum*. O teste de oclusão por *hardware* é realizado se a função retornar que o custo associado à realização do teste é menor que o benefício esperado ou se o nó estiver identificado como previamente ocluso. No cálculo do custo da realização de um teste de oclusão por *hardware* para um nó é também tido em consideração o custo de realizar testes separados aos seus filhos. Por exemplo a um nó com dois filhos, onde um deles se encontra fora do *view-frustum*, obtém-se melhor desempenho testando apenas o nó filho que se encontra dentro do *view-frustum* do que aplicar o teste ao nó pai.

A função que calcula o custo/benefício envolve a estimação do tempo que levaria a fazer directamente o *render* de um nó H_i em comparação com o tempo de realização de um teste de oclusão por *hardware*, mais o tempo que levaria a fazer o *render* caso o nó fosse determinado como visível. De forma semelhante ao algoritmo CHC, são sempre realizados testes aos nós classificados como oclusos na *frame* anterior. A um nó anteriormente visível o teste é realizado se, depois de desenhado durante *n frames* sem lhe ter sido aplicado o teste de oclusão por *hardware*, o custo $C(H_i)$ da realização do teste for menor que o benefício $nB(H_i)$ de uma possível oclusão.

O custo $C(H_i)$ é calculado para cada nível hierárquico separadamente enquanto o benefício $B(H_i)$ é acumulado ao longo de todos os níveis. Desta forma tem que ser

distribuído por todos os níveis dividindo-o pela profundidade da hierarquia. Considerando o número total de nós na hierarquia N_h, obtém-se:

$$C(H_i) = t_0(H_i)$$
$$B(H_i) = \frac{p_0(H_i)[t_r(H_i) - t_0(H_i)]}{log_2(N_h + 1)}$$

Onde:

- $t_0(H_i)$, tempo estimado para a realização do teste de oclusão por *hardware*;
- $p_0(H_i)$, probabilidade de oclusão;
- $t_r(H_i)$, tempo estimado para o *render* do nó;

Os parâmetros t_0 e t_r são obtidos no arranque da aplicação visto serem dependentes do *hardware*. A probabilidade p_0 é determinada com base na porção do ecrã que já foi desenhada por nós anteriores (localizados à frente do nó actual dado o varrimento ser realizado da frente para trás) e na dimensão do nó actualmente em processamento. De forma resumida, a probabilidade de oclusão é considerada alta se muita geometria já tiver sido desenhada e o nó actual for de pequena dimensão. A coerência temporal também é incorporada na equação considerando o estado de visibilidade do nó na *frame* anterior. O método detalhado de determinação da probabilidade p_0 encontra-se descrito em [GUT06].

O algoritmo NOHC é certamente mais complexo que os algoritmos descritos nas secções anteriores. Os autores comparam o seu desempenho com o algoritmo CHC e para isso usam os mesmos modelos e percursos que em [BIT04]. As simulações realizadas mostram uma nítida melhoria no desempenho relativamente ao CHC e ao contrário deste último, nunca mostra um desempenho inferior ao uso do simples *view-frustum culling*, mesmo em cenas com pouca geometria em profundidade. Uma desvantagem deste algoritmo é a necessidade de uma longa etapa de pré-processamento onde são medidos diversos parâmetros dependentes do *hardware* necessários para a função custo/benefício. Esta dependência do *hardware* torna difícil prever o real ganho do algoritmo em diferentes plataformas.

4.5. CHC++

Recentemente, o algoritmo CHC foi revisto por Mattausch *et al.* [MAT08] e adquiriu uma nova designação, CHC++. Com este algoritmo os autores pretendem melhorar diversos aspectos do original, nomeadamente: diminuir as mudanças de estado, reduzir o número de testes de oclusão por *hardware* realizados, diminuir a probabilidade de CPU *stalls*, redução da quantidade de geometria desenhada e a fácil integração em motores de jogos.

Para reduzir as mudanças de estado, o algoritmo usa um mecanismo que permite efectuar conjuntos de testes de oclusão por *hardware* em vez de os realizar isoladamente a cada nó à medida que assim o seja decidido pelo algoritmo. Desta forma, o estado de *rendering* apenas é alterado uma vez por cada conjunto de testes realizados. Os nós oclusos na *frame* anterior são colocados numa fila (*fila-i*). Quando nesta se encontra um número pré-definido de nós, é-lhes aplicado o teste de oclusão por *hardware*. Para nós previamente visíveis, no algoritmo CHC era realizado o teste de oclusão por *hardware* e em seguida prosseguia-se com o *render* da geometria sem esperar pelo resultado dos testes. Da mesma forma, o CHC++ também efectua o render dos nós determinados como visíveis mas a realização dos testes de oclusão por *hardware* não é imediata. Em vez disso, os nós são ordenados e colocados numa outra fila (*fila-v*). Uma visão geral das filas usadas no algoritmo CHC++ é apresentada na Figura 26.

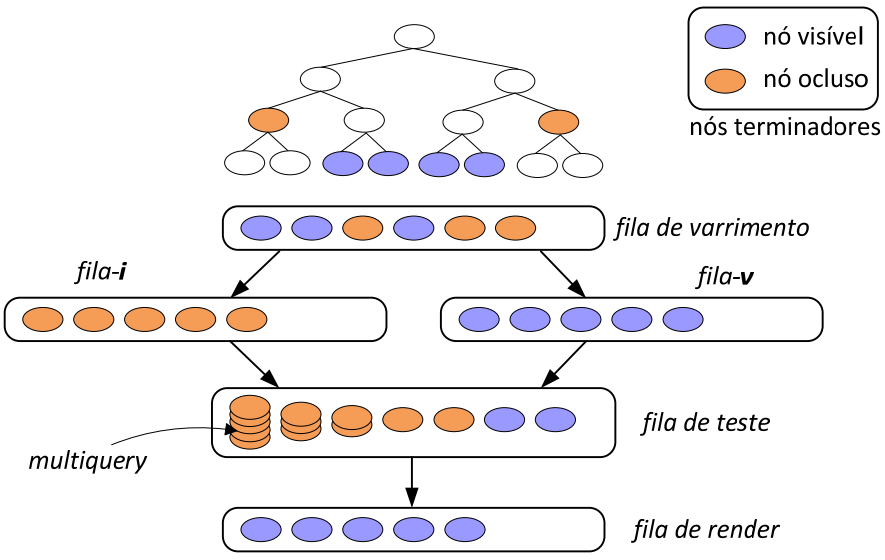


Figura 26 - Esquema de filas usadas no algoritmo CHC++.

Os nós presentes na *fila-v* não são críticos para a *frame* actual já que o resultado do teste de oclusão por *hardware* só será usado na *frame* seguinte. Esta situação é explorada para preencher tempos de espera. Sempre que a *fila de varrimento* se encontra vazia e não existem resultados de testes disponíveis são processados os nós da *fila-v* usando o mesmo método de realização de testes em conjunto.

Para diminuir o número de testes de oclusão por *hardware* realizados em cada *frame*, o algoritmo implementa dois métodos. O primeiro consiste na realização de um único teste de oclusão por *hardware* que engloba vários nós oclusos, ao qual é dado o nome de *multiquery*. Se uma parte da cena previamente oclusa mantém o seu estado de visibilidade na *frame* actual, um único teste para essa parte será necessário. Este teste é realizado usando um volume envolvente que engloba todos os objectos em causa. No entanto, se o teste de oclusão por *hardware* revelar que esta parte da cena se encontra visível são realizados novos testes para todos os nós do grupo inserindo-os novamente na *fila-i*. Para encontrar grupos de objectos candidatos a constituírem uma mesma *multiquery* é usada uma função custo/benefício.

Uma das parcelas usadas no cálculo do custo é a probabilidade (p_{keep}) de um nó manter o seu estado de visibilidade na *frame* seguinte. Experiências realizadas pelos autores demonstram uma forte correlação entre esta probabilidade e o histórico do nó, *i.e.*, o número de *frames* que o nó manteve o mesmo estado de visibilidade. A isto os autores dão o nome de *persistência de visibilidade* (*i*). No entanto, a aplicação deste método é prejudicado pelo facto de no início da simulação não existirem dados suficientes para determinar com precisão a probabilidade necessária ao cálculo do custo. Uma alternativa proposta consiste no uso de uma função analítica que corresponde razoavelmente bem à função de valores medidos pelo método anterior:

$$p_{keep}(i) \approx 0{,}99 - 0{,}7e^{-i}$$

Uma abordagem semelhante ao algoritmo CHC já tinha sido proposta por Kovalcík e Sochor [KOV05] em 2005.

O segundo método para diminuir o número de testes de oclusão por *hardware* aplica-se a nós visíveis na *frame* anterior. No algoritmo CHC assume-se que um nó fica visível por *n*

frames sendo apenas testado na *frame n*+1. Esta aplicação da coerência temporal reduz o número médio de testes mas origina o alinhamento temporalmente dos mesmos. Esta situação pode ser problemática se muitos nós ficarem visíveis na mesma *frame*. Por exemplo quando, numa cena urbana, se sobe ao topo de um edifício. Para minimizar este efeito é realizada a distribuição dos testes de oclusão por *hardware* a nós visíveis de forma aleatória através de um intervalo de *frames*. Este método evita o alinhamento e a propagação temporal dos testes de oclusão por *hardware* verificados no CHC devido ao número fixo de *frames* usado como intervalo entre testes.

Este algoritmo revela um desempenho superior ao NOHC, CHC e *view-frustum culling* em todas as situações. De todas as alterações implementadas, a realização de testes em conjunto foi a que produziu maior aumento relativo de desempenho. No entanto, o facto de ser necessário aguardar que um número pré-definido de nós esteja disponível nas filas para realizar os testes leva a um atraso na disponibilidade dos resultados. Isto significa que alterações na visibilidade podem ser detectadas mais tarde do que seria desejável. Além do mais, na situação de um nó continuar a ser considerado ocluso, mais testes seriam realizados sobre ele o que aumentaria ainda mais a latência. As *multiqueries* permitem reduzir o número de testes de forma equivalente ao número total de objectos usados no grupo testado. Se a *multiquery* retornar que o volume envolvente testado é visível, este teste torna-se desnecessário e todos os objectos são novamente testados de forma individual.

4.6. SUMÁRIO

Os algoritmos que usam testes de oclusão por *hardware* são atractivos para determinar a visibilidade durante a execução da aplicação devido ao seu princípio de funcionamento simples. É realizado um teste. Se este falhar, a geometria pode ser removida de posteriores processamentos. O principal problema a ter em consideração no seu uso é a latência entre a realização do teste e a disponibilidade do resultado. Se esta latência não existisse, os testes de oclusão por *hardware* seriam o método ideal para solucionar o problema da remoção por oclusão.

As abordagens apresentadas neste capítulo tentam, com diferentes graus de complexidade, minimizar os efeitos negativos associados ao uso dos testes de oclusão por *hardware*. Para evitar testar toda a geometria presente no *view-frustum*, os algoritmos recorrem a diversas técnicas como: identificar o conjunto mínimo de nós que necessitam de ser testados em cada *frame*, aplicar o princípio da coerência temporal, aumentar a geometria usada em cada teste ou aplicar uma função custo/benefício que se adapta a diferentes condições da cena. Assim, o objectivo comum a todas estas técnicas é o de reduzir o número de testes de oclusão por *hardware* realizados por *frame* mantendo a disponibilidade de informação necessária para determinar a visibilidade com precisão. A Tabela 1 apresenta a comparação dos algoritmos descritos no presente capítulo.

Tabela 1 – Comparação de algoritmos que usam testes de oclusão por *hardware*.

	Determinação de visibilidade	**Espaço onde operam**	**Organização da cena 3D**	**Pré-processamento**	**Coerência temporal**	**Parâmetros ajustáveis**	**Teste de oclusão a nós oclusos**	**Teste de oclusão a nós visíveis**
cPLP	ponto	objectos e imagem	*octree*	cálculo da "solidez" dos nós	não	orçamento de polígonos	todas as *frames*	todas as *frames*
FSOC	ponto	imagem	grelha uniforme	criação da grelha	não	resolução da grelha, número de testes a realizar em sequência	todas as *frames*	todas as *frames*
CHC	ponto	imagem	*kD-tree*	-	sim	intervalo entre testes a nós visíveis	todas as *frames*	intervalo de *frames*, se forem nós terminadores
NOHC	ponto	imagem	p-HBVO [MEI99]	recolha de parâmetros de desempenho associados ao *hardware*	sim	-	todas as *frames*	intervalo de *frames*, função custo/benefício
CHC++	ponto	imagem	Hierarquia de Volumes Envolventes, alinhados com os eixos	-	sim	nº de nós necessários para uma *multiquery*, intervalo entre testes a nós visíveis	todas as *frames* (em conjuntos, *multiquery*)	intervalo de *frames*, se forem nós terminadores

5. OPENSCENEGRAPH

O presente capítulo destina-se a apresentar a API que disponibiliza as funcionalidades de grafos de cena e de *rendering* usadas na aplicação desenvolvida no contexto desta dissertação. É também apresentada e analisada a sua actual implementação do algoritmo de remoção que usa testes de oclusão por *hardware*, o qual será a base do algoritmo proposto no capítulo seguinte.

O OpenSceneGraph (OSG) é um projecto *open source* com o objectivo de criar um conjunto de ferramentas para o desenvolvimento de aplicações gráficas de alto desempenho multi-plataforma [OSG]. A iniciativa remonta aos finais dos anos 90 e actualmente conta com uma vasta comunidade de utilizadores e programadores bastante activa. Em [MAR07] é feita a introdução histórica, descrição da arquitectura, principais características e funcionalidades do OSG. Disponível gratuitamente na sua versão electrónica, este livro é um bom ponto de partida para quem pretenda começar a desenvolver aplicações baseadas no OSG.

Na sua essência, o OSG baseia-se no conceito de grafos de cena e adopta uma arquitectura Orientada a Objectos tendo o OpenGL como base. Escrito inteiramente em ANSI C++ e OpenGL, usa *Standard Template Library* [STL99] e *Design Patterns* [GAM95]

para disponibilizar bibliotecas de funcionalidades que permitem o desenvolvimento de aplicações 3D de alto desempenho. O resultado é uma API que disponibiliza os benefícios dos grafos de cena a todo o tipo de utilizadores, empresariais e particulares. O OSG é *middleware*, situando-se na camada intermédia acima da API de baixo nível e abaixo da aplicação 3D. As vantagens apresentadas pelo OSG são o seu desempenho, escalabilidade, portabilidade e o uso de grafos de cena.

5.1. VISÃO GERAL

O OSG é desenhado como uma camada de abstracção de alto nível que disponibiliza uma interface de programação para o desenvolvimento de aplicações gráficas de alto desempenho. O OSG é também concebido para ser flexível e extensível o que permite um desenvolvimento adaptativo ao longo do tempo. Como resultado, o OSG consegue colmatar as expectativas dos utilizadores à medida que estas vão aparecendo. O uso de uma estrutura hierárquica para um *render* eficiente, o emprego de técnicas de gestão de memória e a capacidade para lidar como diversos modelos 3D formam a base que torna o OSG uma boa escolha para o desenvolvimento de aplicações gráficas.

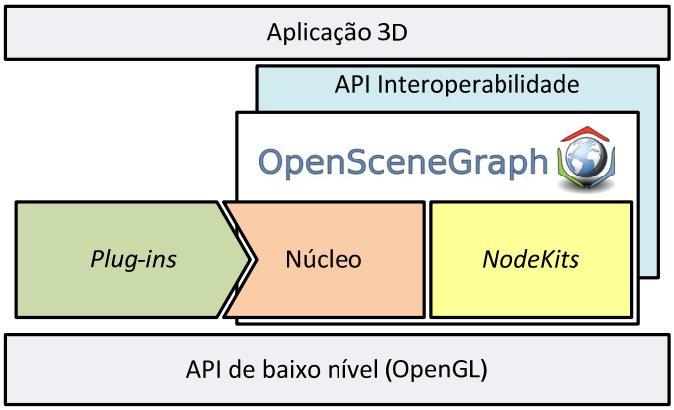


Figura 27 - Arquitectura do OpenSceneGraph e localização relativa às restantes camadas.

Na arquitectura do OSG (Figura 27) o núcleo é o conjunto de bibliotecas que fornecem funcionalidades tanto à aplicação como aos *NodeKits*. Em conjunto, as bibliotecas do núcleo e os *NodeKits* constituem a API do OSG. As bibliotecas nucleares fornecem funcionalidades essenciais de grafos de cena e de *rendering*, para além de funcionalidades adicionais que tipicamente as aplicações 3D necessitam. Os *NodeKits* permitem tipos de nós de mais alto nível para o grafo de cena e também efeitos especiais

(*e.g.*, *bump mapping* e *cartoon shading*). Os *Plug-ins* são bibliotecas que lêem e escrevem imagens 2D e modelos 3D. As bibliotecas de interoperabilidade permitem que o OSG seja facilmente integrado em novos ambientes, incluindo linguagens de *scripting* como o Python[3]e Lua[4].

Dos componentes da arquitectura do OSG enumerados destaca-se o núcleo, composto por quatro bibliotecas:

- **osg** — biblioteca que contém as classes dos nós que a aplicação usa para criar o grafo de cena. Também contém classes para operações com vectores e matrizes, geometria e especificações de estado para o *rendering*. Outras classes da biblioteca fornecem funcionalidades que tipicamente as aplicações 3D necessitam tais como análise de argumentos, animações e comunicação de erros e avisos.

- **osgUtil** — contém classes e funções para realizar operações no grafo de cena e seu conteúdo, recolha de estatísticas, optimização do grafo de cena e criação do grafo para *rendering*. Também contém classes para operações geométricas, tais como a triangulação de Delaunay, ou a geração de coordenadas de texturas.

- **osgDB** — disponibiliza classes e funções para criação e *render* de bases de dados 3D. Contém um registo dos *Plug-ins* do OSG para leitura/escrita de ficheiros 2D e 3D bem como uma classe para aceder a esses *Plug-ins*. O *database pager* da biblioteca suporta o carregamento e descarregamento dinâmico de grandes segmentos da base de dados da cena 3D.

- **osgViewer** — esta biblioteca contém classes que gerem vistas sobre a cena e que permitem a integração do OSG com uma larga variedade de sistemas de janelas. Também disponibiliza classes de suporte para recolha de estatísticas e manipulação da cena. Todas as classes desta biblioteca estão preparadas para aplicações *multi-thread*.

[3] http://www.python.org

[4] http://www.lua.org

Das bibliotecas mencionadas, a mais relevante no contexto da criação de um grafo de cena é a biblioteca **osg** que fornece as classes que definem os nós elementares que constituem um grafo. Todas as classes dos diferentes tipos de nós são derivadas da classe **osg::Node** apesar de conceptualmente, nós raiz, grupo e folha serem tipos de nós distintos. Com esta abordagem orientada aos objectos, o OSG disponibiliza uma variedade de tipos de nós onde a sua habilidade na organização espacial permite capacidades de armazenamento de informação que não estão disponíveis nas tradicionais APIs de baixo nível. Todos os nós partilham a classe comum (**osg::Node**) com as funcionalidades especializadas de cada nó definidas nas classes derivadas. Algumas dessas classes são:

- **osg::Node** – é a classe base para todos os nós no grafo de cena. Contém métodos para varrimento do grafo, remoção, *Callbacks* e gestão de estado.

- **osg::Group** — é a classe base para qualquer nó que possa ter nós filho, desempenha um papel chave na organização espacial dos grafos de cena.

- **osg::Geode** — esta classe corresponde ao nó folha no OSG. Contém a geometria a desenhar (objectos do tipo **osg::Drawable**) e não tem filhos.

- **osg::LOD** (*Level Of Detail*)— esta classe activa os seu filhos com base na sua distância ao ponto de vista actual. É usado para mostrar vários níveis de detalhe de objectos numa cena.

- **osg::MatrixTransform** — transforma a geometria dos seus filhos, permitindo que objectos da cena sejam rodados, transladados, escalados, projectados, entre outras operações.

Esta é uma lista incompleta dos tipos de nós suportados pelo OSG, existem outros como o **osg::Switch** , **osg::Sequence**, **osg::Transform**, **osg::LightSource**, **osg::Billboard** e **osg::OcclusionQueryNode**. Este último, é a essência do mecanismo de remoção por oclusão implementado no OSG. Como tal, na secção seguinte, é objecto de estudo mais aprofundado.

5.2. TESTE DE OCLUSÃO POR *HARDWARE* NO OPENSCENEGRAPH

A biblioteca nuclear **osg** contém as classes dos nós usados para a criação do grafo de cena no OSG. Uma dessas classes, define um nó que pode ter filhos, **osg::Group**. Das várias classes derivadas ilustradas na Figura 28, destaca-se a classe **osg::OcclusionQueryNode**.

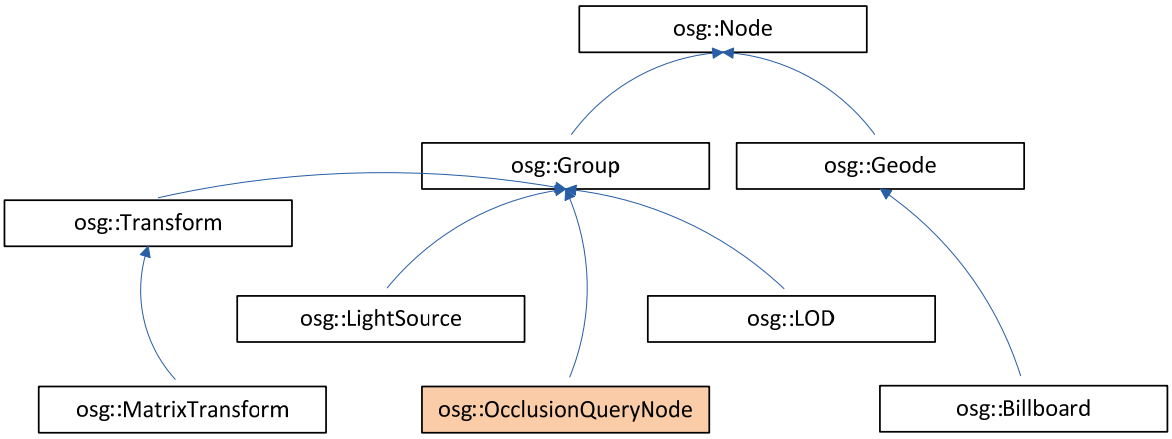


Figura 28 - Relação entre algumas das classes de diferentes tipos de nós presentes na biblioteca osg.

O nó implementado por esta classe, doravante denominado por **nó OQ**, permite realizar o teste de oclusão por *hardware* (Secção 3.6) a todos os seus nós filhos. Para isso usa o correspondente volume envolvente e disponibiliza posteriormente o estado de visibilidade da geometria à aplicação. A classe que implementa o nó OQ disponibiliza métodos que permitem ligar/desligar o uso do nó e definir o número de píxeis mínimo para que o volume envolvente testado seja considerado visível (quando comparado com o número de píxeis visíveis obtido do teste de oclusão por *hardware*). Também permite estabelecer um intervalo de *frames* entre a realização de testes de oclusão por *hardware* ao nó. Estas funcionalidades permitem que a aplicação faça uma abordagem conservadora ou aproximada à determinação de visibilidade e que explore a coerência temporal. Um nó OQ inactivo presente num grafo de cena comporta-se como um normal nó grupo.

Uma aplicação gráfica que use a API do OSG e que pretenda usar o mecanismo de testes de oclusão por *hardware* para aferir a visibilidade da geometria a enviar para *rendering*, pode fazer uso dos nós OQ de duas formas. A primeira consiste em aplicar os nós em *runtime*, por exemplo, apenas à porção da cena que foi identificada como se encontrando

dentro do *view-frustum* numa dada *frame*. A outra forma é inserir permanentemente os nós OQ na estrutura hierárquica da cena.

O teste de oclusão por *hardware* no OSG ao ser implementado como um nó, permite que numa etapa de pré-processamento o grafo de cena seja analisado e, com base em determinados critérios, inserir nós OQ na estrutura hierárquica. Um critério de decisão pode ser, por exemplo a quantidade de primitivas contidas num nó (incluindo todos os nós filhos). Se este número for inferior a um limite predefinido não é inserido um nó OQ como pai do nó em causa. Neste caso, o custo de desenhar a geometria quando comparado com o custo do teste de oclusão por *hardware* não justifica o seu uso. Durante o processamento de uma *frame*, o algoritmo que decide a realização de testes de oclusão por *hardware* apenas irá abranger os nós OQ inseridos e assim evitar desperdício de recursos ao testar geometria desnecessariamente.

Durante a etapa de Remoção (Secção 2.2), o grafo de cena é percorrido para seleccionar a geometria a desenhar na presente *frame*, o que significa realizar o *view-frustum culling* e a remoção por oclusão. É nesta etapa que são processados os nós OQ. Durante este processamento, três situações podem ocorrer:

- O nó OQ e a sua árvore filha são removidos de posteriores processamentos por se encontrarem fora do *view-frustum*.

- Durante a *frame* actual, se o mais recente resultado do teste de oclusão por *hardware* indicar que o nó se encontra ocluso, é realizado novo teste e a árvore filha do nó não é processada. No entanto, o teste de oclusão por *hardware* só é realizado se o número de *frames* de intervalo entre testes, definido para o nó em questão, tiver sido excedido.

- A terceira situação ocorre quando o nó em processamento é considerado visível. Para que isso aconteça, o resultado do teste de oclusão por *hardware* (nº de píxeis visíveis) terá que ser superior a um valor predefinido de píxeis. Este valor decide o limite de visibilidade para o nó em questão. Neste caso, o processamento prossegue para os nós filhos. O nó também é considerado visível se o actual ponto de vista se encontrar dentro da sua esfera envolvente. Para actualizar a

informação de visibilidade é realizado novo teste de oclusão por *hardware* segundo os mesmo critérios que para os nós oclusos (intervalo de *frames*).

A Figura 29 ilustra o comportamento descrito. O algoritmo que determina a visibilidade do nó OQ é aplicado durante a etapa de Remoção durante a qual a estrutura hierárquica é varrida em profundidade. As etapas a tracejado pertencem a métodos da classe **osg::OcclusionQueryNode**.

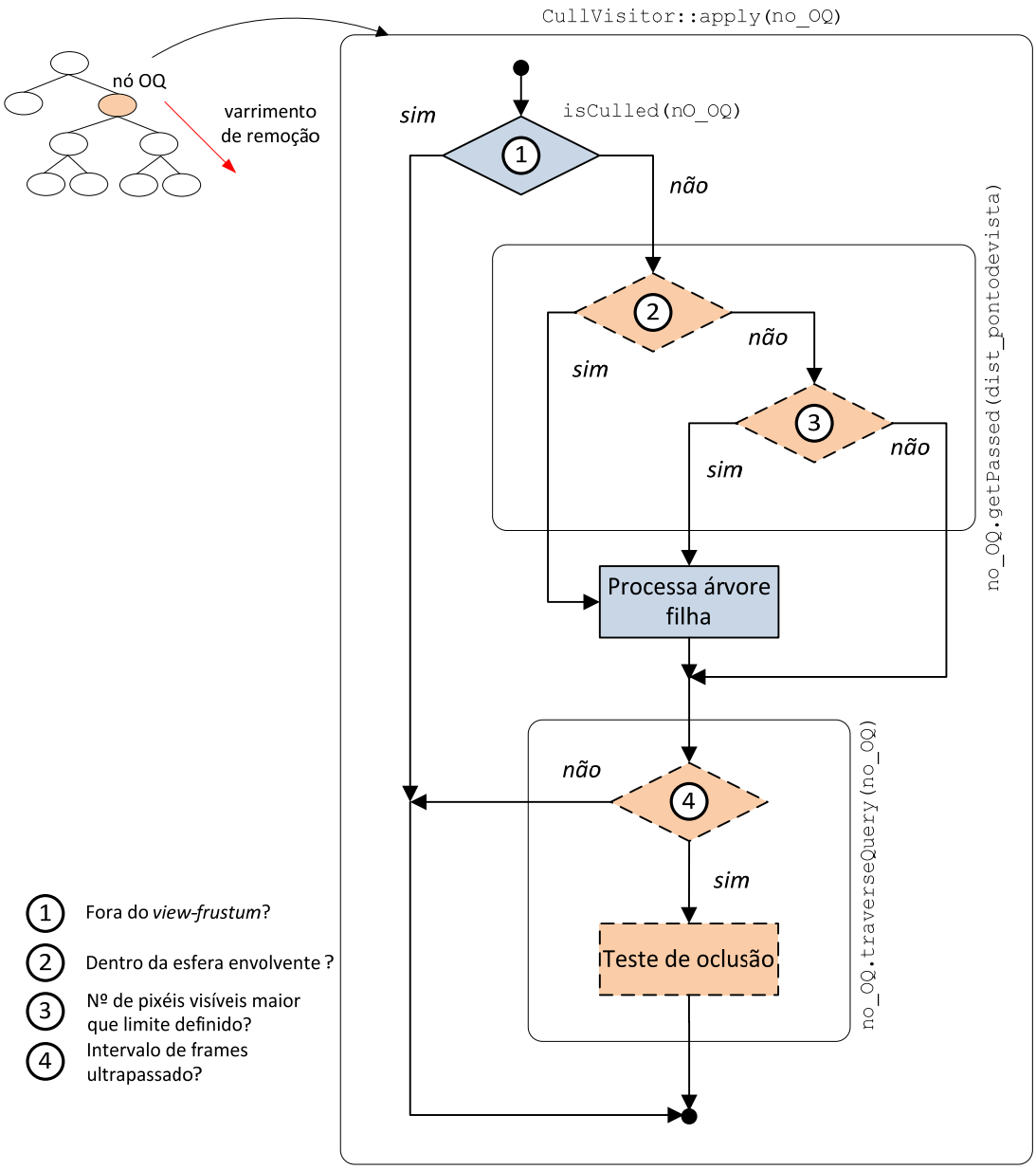


Figura 29 - Algoritmo de remoção por oclusão do OSG.

5.2.1. ANÁLISE

Quando comparado com alguns dos algoritmos apresentados no capítulo anterior, o algoritmo de remoção por oclusão implementado na actual versão do OSG é de certa forma mais simples. Durante o varrimento de Remoção, os nós OQ são processados de forma semelhante, sejam estes identificados como oclusos ou como visíveis na *frame* anterior. A única diferença reside no facto de no caso de serem visíveis o algoritmo avançar com o processamento dos nós filhos. No caso de estes serem nós folha, significa realizar o seu *render* na *frame* actual. Outros algoritmos atribuem prioridades ou aplicam tratamentos diferentes a estes dois tipos de nós com o objectivo de gerir o número de testes realizados em conjunto com a relevância que cada nó terá para a imagem final.

A abordagem de realizar um novo teste de oclusão por *hardware* apenas quando um intervalo predefinido de *frames* (configurável para cada nó OQ) tenha decorrido, origina um menor número de testes relativamente a algoritmos que realizam um novo teste sempre que um nó é dado como ocluso na *frame* anterior. Esta aplicação do princípio da coerência temporal significa que o algoritmo assume que um nó, independentemente do seu estado de visibilidade, tende a permanecer no mesmo estado nas *frames* seguintes. Por outro lado, este intervalo de *frames* implica que a informação para a correcta determinação de visibilidade possa estar desactualizada na *frame* actual. Como consequência, a variação deste intervalo de *frames* terá uma enorme influência na geometria desenhada:

- Para grandes intervalos os aspectos negativos dos testes de oclusão por *hardware* têm menor influência na aplicação o que permite, em princípio, um melhor desempenho. No entanto, a quantidade de geometria erradamente desenhada aumenta. Por exemplo quando um objecto, que na realidade se encontra visível, não é desenhado devido a continuar a ser considerado ocluso em consequência da falta de testes de oclusão por *hardware* actualizados. Neste caso, muita geometria poderá ser desenhada quando na realidade poderia estar oclusa pelo objecto que não foi desenhado por continuar a ser erradamente considerado ocluso. Esta situação é elucidativa de que nem sempre a diminuição da frequência de testes de

oclusão por *hardware* implica uma diminuição da geometria desenhada e consequentemente um melhor desempenho.

- Para pequenos intervalos, a disponibilidade de informação actualizada para determinar correctamente a visibilidade permite diminuir a quantidade de geometria erradamente desenhada. O pior caso será a realização de testes em todas as *frames*. No entanto, o aumento da frequência de testes de oclusão por *hardware* pode levar a uma degradação no desempenho da aplicação devido à acumulação do custo associado a cada teste.

Outro problema que advém do intervalo pré-definido de *frames* é o alinhamento temporal de novos testes de oclusão por *hardware*. Enquanto no algoritmo CHC esse alinhamento ocorre nos testes realizados aos nós visíveis, neste caso, também acontece nos testes aplicados aos nós oclusos. Esta situação é mais penalizadora quando uma grande quantidade de nós se torna visível/ocluso num curto espaço de tempo. Em simulações em tempo real com *frame rate* variável, ou simulações que não consigam manter uma *frame rate* constate predefinida, o uso de um intervalo fixo de *frames* entre testes de oclusão por *hardware* implica uma variação na disponibilidade de resultados e consequentemente na correcção da imagem final.

O nó OQ permite definir o número de píxeis a partir do qual o volume envolvente testado é considerado visível. Assim, a variação deste parâmetro permite regular a agressividade do algoritmo na determinação do conjunto potencialmente visível. Pequenos valores significam uma abordagem mais conservadora onde geometria que na realidade se encontra oclusa é considerada visível devido à natureza dos volumes envolventes.

Os dois parâmetros descritos podem ser ajustados de forma individual para cada nó OQ. Isto permite que a aplicação ajuste os valores para resolver situações particulares ou definir comportamentos específicos para determinadas zonas/objectos da cena. No entanto, o uso mais comum é a definição dos parâmetros de forma global para a todos os nós OQ. Esta forma de operar não inviabiliza a alteração dos parâmetros em *runtime* para que, por exemplo, a aplicação consiga manter uma *frame rate* constante em detrimento de uma imagem mais correcta.

5.3. SUMÁRIO

O projecto OpenSceneGraph apresentado neste capítulo disponibiliza ferramentas que permitem o desenvolvimento de aplicações gráficas de alto desempenho. Das bibliotecas que constituem o núcleo da API destaca-se a que define as classes que implementam os nós que são usados para criar grafos de cena, a biblioteca **osg**. Um dos vários nós disponíveis, o **osg::OcclusionQueryNode**, permite realizar o teste de oclusão por *hardware* à sua árvore filha. Os parâmetros configuráveis deste tipo de nó permitem adoptar uma abordagem mais conservadora ou mais agressiva na determinação de visibilidade. O algoritmo de remoção por oclusão implementado pelo OSG é de certa forma mais simples que os algoritmos apresentados no Capítulo 4. No entanto, este algoritmo constitui uma boa base de trabalho para o estudo da influência que a aplicação hierárquica de testes de oclusão por *hardware* tem no desempenho da simulação de cenas com diferentes níveis de complexidade.

6. REMOÇÃO HIERÁRQUICA POR OCLUSÃO

Nos capítulos anteriores foram introduzidos conceitos e apresentadas algumas das técnicas actualmente adoptadas na determinação de visibilidade com incidência nas que se destinam a remover geometria por oclusão. Estas técnicas quando aplicadas a cenas 3D organizadas numa estrutura de dados hierárquica permitem remover grandes quantidades de geometria com poucos cálculos. De uma forma simples, se um nó num nível superior da hierárquica for identificado como ocluso toda a sua árvore filha poderá ser removida de posteriores processamentos diminuindo a quantidade de geometria total a ser processada em cada *frame*.

No presente capítulo propõe-se uma alternativa de implementação de remoção por oclusão no OSG que tira partido da organização hierárquica dos testes de oclusão por *hardware* e da coerência temporal. Apresentam-se também os resultados de diversas simulações onde o teste de oclusão por *hardware* foi aplicado em diferentes níveis hierárquicos do grafo de cena. Os resultados são analisados na perspectiva de analisar em que medida os testes influenciam o desempenho da aplicação. Nas simulações foram usadas cenas de muito grandes dimensões com diferentes níveis de complexidade

geométrica. Todas as simulações foram realizadas numa aplicação desenvolvida para o efeito que usa a API do OSG para funcionalidades de grafo de cena e de *rendering*.

6.1. Influência da Hierarquia na Remoção por Oclusão

Os algoritmos de remoção por oclusão descritos no Capítulo 4 pretendem ser genéricos e aplicáveis a cenas genéricas. Estes algoritmos usam estruturas hierárquicas para organizar as cenas e ao aplicarem os testes de oclusão por *hardware* usufruem das vantagens proporcionadas por este tipo de organização espacial. No entanto, nos artigos onde é feita a apresentação dos diversos algoritmos não é estudado o impacto no desempenho da aplicação que a realização de testes de oclusão por *hardware* teria se fossem aplicados apenas em alguns níveis hierárquicos, ou combinações destes. Nesta secção é feito um levantamento de como a forma da estrutura hierárquica, neste caso os grafos de cena, e a aplicação dos testes de oclusão por *hardware* a esta podem influenciar a quantidade de geometria a processar.

A forma como os nós são organizados num grafo de cena tem uma importância vital no desempenho dos vários algoritmos que necessitam de percorrer a estrutura hierárquica, seja para actualizar e recolher informação ou realizar qualquer outro tipo de processamento, como o *view-frustum culling*. Uma cena pode ser representada por vários grafos de cena que reflectem diferentes formas de organizar os mesmos objectos no espaço 3D. A Figura 30 ilustra uma cena exemplo e dois possíveis grafos de cena que a representam.

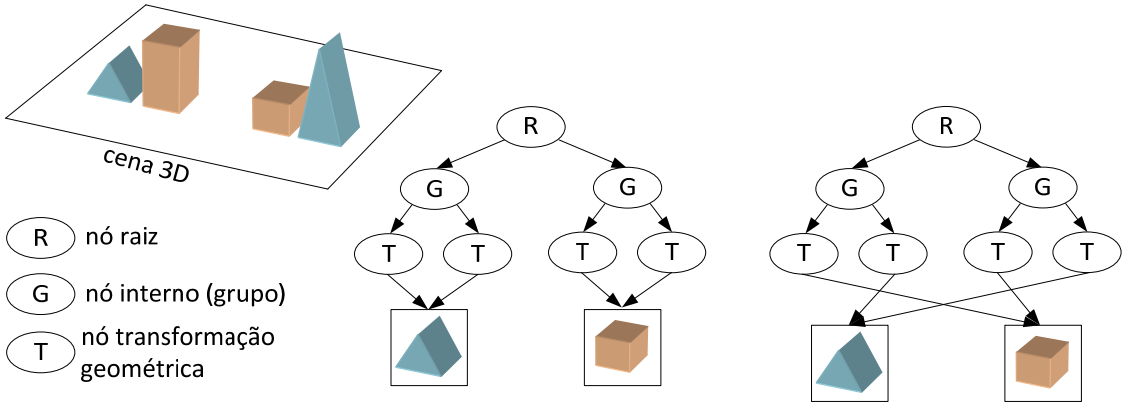


Figura 30 – A mesma cena 3D representada por dois grafos de cena diferentes.

No caso da Figura 30, o grafo de cena mais adequado é o da direita pois incorpora a coerência no espaço dos objectos o que permite uma remoção mais eficiente. Além desta situação, para que o grafo de cena esteja afinado para a operação de remoção cada nó não deverá ter nem muitos nem poucos nós filho, *i.e.*, não deverá ser nem demasiado profundo nem demasiado plano. Nesta perspectiva, a construção de grafos de cena balanceados oferece o melhor compromisso para o custo de processamento da estrutura.

Um grafo de cena apenas com um nó (raiz) diz-se ter uma altura (*h*) de 0. Um nó filho do nó raiz encontra-se à altura 1 e assim sucessivamente. Num grafo balanceado todos os nós folha encontram-se à altura *h* ou *h*-1. Em geral, a altura *h* de um grafo balanceado é $\log_k n$, onde *n* é o número total de nós (internos e folha) e *k* o número de filhos de cada nó interno [AKE08]. Por exemplo, um *k* elevado origina uma estrutura hierárquica menos profunda o que significa menos passos para varrer a árvore, mas também requer mais processamento em cada nó.

Os algoritmos de remoção por oclusão são usados durante a etapa de Remoção para seleccionar a geometria a excluir da imagem final. Durante esta etapa, o grafo de cena é percorrido em profundidade começando pelo nó raiz da hierarquia, sendo a visibilidade de cada nó determinada aplicando o algoritmo. Se um nó é considerado ocluso, toda a sua árvore filha é removida, *i.e.*, não são realizados testes de oclusão por *hardware* e não é desenhada a correspondente geometria. Se o nó for considerado visível, os seus descendentes são testados para apurar o conjunto de nós visíveis. Assim, a remoção hierárquica por oclusão deverá representar ganhos devido a evitar testes de oclusão por *hardware* desnecessários e a remover geometria que não contribui para a imagem a apresentar ao utilizador.

A quantidade de níveis hierárquicos usados na representação de uma cena influencia a determinação de visibilidade na medida em que, para uma mesma quantidade de nós folha (geometria), o uso de vários níveis hierárquicos implica que cada um dos nós internos tem menos filhos. Desta forma, a quantidade de geometria abrangida por cada teste de oclusão por *hardware* aplicado a nós internos varia com a profundidade da estrutura hierárquica.

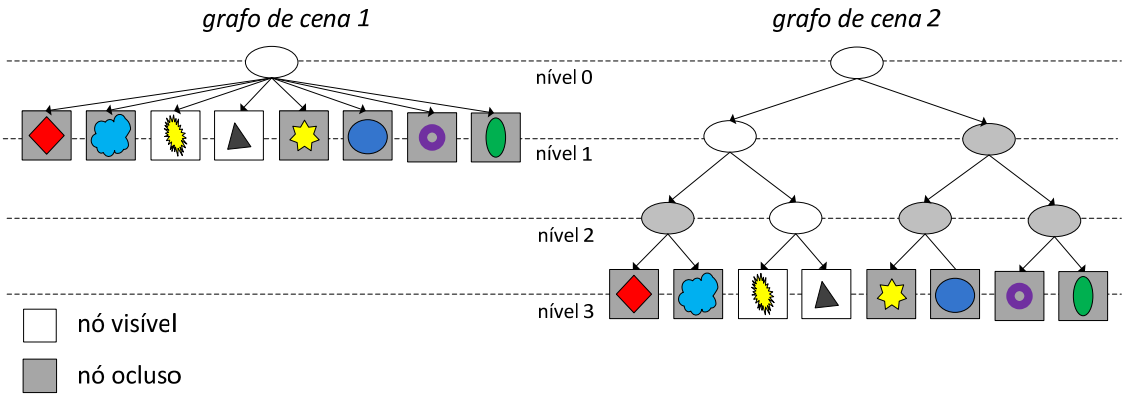


Figura 31 – Grafos de cena com diferentes níveis hierárquicos que organizam os mesmos objectos.

Para determinar a visibilidade dos nós da estrutura hierárquica plana apresentada à esquerda na Figura 31, considere a aplicação do algoritmo de remoção apenas ao nó raiz. O volume que envolve todos os nós filhos é usado como geometria para o teste de oclusão por *hardware*. Nesta situação, se o teste revelar que o volume envolvente se encontra ocluso o ganho é significativo pois não é necessário testar todos os nós descendentes. Por outro lado, nós que contêm muitos objectos abrangem uma grande zona da cena o que torna mais difícil que se encontrem totalmente obstruídos da vista por outros objectos da cena. Esta situação é ainda mais problemática em cenas com baixa densidade de geometria. A aplicação do teste de oclusão por *hardware* apenas aos nós folha implica um maior número de testes já que todos os nós são sempre testados devido à ausência de um meio que permita remover vários objectos com um único teste. Neste caso, também cenas com baixa densidade originam testes desnecessários no sentido em que grande parte da geometria testada é determinada como visível. Em cenas de muito grandes dimensões uma organização hierárquica deste tipo não seria eficiente.

O grafo de cena à direita na Figura 31 representa uma organização da cena com vários níveis hierárquicos. Se os testes de oclusão por *hardware* forem apenas aplicados ao nó raiz ou aos nós folha o comportamento é igual ao descrito anteriormente. No entanto, a existência de nós intermédios que abrangem menos geometria que o nó raiz permite o teste de volumes envolventes que abrangem zonas mais pequenas da cena. Isto assumindo que o grafo de cena tem em consideração a localização espacial, agrupando objectos que se encontram próximos. Ao usar volumes envolventes menores, a probabilidade de estes se encontrarem oclusos aumenta o que permite a remoção de

toda a sua árvore filha. Em cenas de muito grandes dimensões, a realização de testes a nós grupo intermédios permite remover grandes quantidades de geometria com um único teste. No entanto se estes falharem, testes aos nós folha permitem resolver o problema da visibilidade dos objectos individualmente. No exemplo da figura, para determinar a visibilidade na situação específica ilustrada são necessários 8 testes de oclusão por *hardware* no *grafo de cena 1* e 6 no *grafo de cena 2*, considerando que o teste é aplicado a todos os níveis hierárquicos, excepto ao nó raiz.

As observações realizadas anteriormente permitem delinear um conjunto de directivas a ter em consideração na aplicação hierárquica de algoritmos de remoção por oclusão:

1. A realização de testes de oclusão por *hardware* ao nível da geometria aumenta significativamente o número de testes realizados mesmo quando usados em conjunto com testes nos níveis hierárquicos superiores.

2. O uso de testes de oclusão por *hardware* apenas em níveis superiores da hierárquica implica que muita geometria será desenhada mesmo que oclusa pela falta de um mecanismo que resolva a visibilidade ao nível dos objectos. A aplicação de testes a estes níveis hierárquicos apenas apresentará vantagens em cenas de grandes dimensões e elevada complexidade.

3. Testes de oclusão por *hardware* aplicados a nós grupo internos permitem eliminar grandes quantidades de geometria com um único teste. No entanto, é sempre necessário ter em consideração o custo caso o nó seja considerado visível.

4. A dimensão do volume envolvente da árvore filha de um nó ao qual é aplicado o teste de oclusão por *hardware* não deverá ser tal que inviabilize que este se torne ocluso durante uma simulação.

5. A aplicação de testes a níveis hierárquicos não adjacentes implica grandes variações no desempenho provocadas pela súbita remoção/inclusão de grandes quantidades de geometria.

6. Em cenas com baixa densidade de geometria é necessária precaução na aplicação dos testes de oclusão por *hardware*. O custo acumulado dos testes pode não compensar o ganho obtido quando a geometria é determinada como oclusa.

7. Em cenas de grandes dimensões e elevada densidade a aplicação de testes aos nós internos em conjunto com o teste ao nível da geometria permite uma remoção com diferentes níveis de resolução o que deverá ser benéfico para o desempenho da aplicação.

Das considerações enunciadas nenhuma coloca em causa a geometria que na realidade contribui para a imagem final, apenas afectam a dimensão do conjunto potencialmente visível determinado em cada *frame* para um particular ponto de vista sobre a cena. A dimensão do volume envolvente face à densidade de geometria na cena é preponderante para a eficiência do algoritmo de remoção por oclusão. A selecção do número de filhos de cada nó e a forma como a geometria é organizada no grafo de cena, devem ter em consideração a dimensão e localização (coerência espacial) dos objectos para uma remoção mais eficiente.

6.2. APLICAÇÃO DESENVOLVIDA

Para comprovar a influência prevista da hierarquia na remoção por oclusão e avaliar a sua utilização mais eficiente foi desenvolvida uma aplicação para realizar as simulações necessárias. A aplicação 3D foi implementada em C++ fazendo uso da API do OSG. Para além da estrutura e funcionalidades base necessárias a simulações visuais em tempo real, a aplicação inclui funcionalidades específicas para o trabalho em causa. Algumas destas funcionalidades implicaram a modificação da própria API do OSG. Uma destas modificações foi a edição das classes de alguns tipos de nós para permitir a implementação de uma abordagem alternativa ao algoritmo original de remoção por oclusão do OSG. Com a actual distribuição do OSG, é fornecida uma grande quantidade de aplicações exemplo que ilustram conceitos de programação e técnicas usadas no desenvolvimento de aplicações OSG. Alguns destes exemplos revelaram-se bastante úteis durante o processo de implementação da aplicação ao permitirem um mais rápido entendimento do uso de determinadas chamadas à API.

6.2.1. VISÃO GERAL DAS FUNCIONALIDADES DA APLICAÇÃO

A classe *Viewer* da biblioteca **osgViewer** é usada para criar a janela da aplicação e o contexto gráfico com base nas capacidades do sistema gráfico em uso. Esta classe em conjunto com a classe *ReadFile* da biblioteca **osgDB**, permite criar uma aplicação simples que lê geometria a partir de ficheiros de modelos 3D e a apresenta numa janela. Dependendo dos *Plug-ins* disponíveis é possível utilizar uma variada gama de formatos de imagens e modelos 3D. A Figura 32 apresenta a aplicação, com a visualização de estatísticas no ecrã activada, a mostrar um dos modelos 3D (*cow.osg*) fornecidos com o OSG.

Figura 32 - Aspecto geral da aplicação desenvolvia.

Várias funcionalidades foram adicionadas à aplicação recorrendo à API do OSG: análise de argumentos de linha de comando, gestão de eventos (teclado e rato), manipulador da câmara genérico, captura de imagens do ecrã, visualização da posição da câmara e de estatísticas no ecrã, entre outras. Para atingir os objectivos a que se destina a aplicação foram implementadas funcionalidades específicas:

- Criação de grafos de cena: esta funcionalidade permite, com base em dados fornecidos pelo utilizador, construir grafos de cena onde é possível definir diversos parâmetros. Por exemplo, o número de nós de geometria que devem ser inseridos como filhos de um nó grupo. Este processo é apresentado em maior detalhe na Secção 6.3 que descreve as cenas criadas para as simulações.

- Importação e exportação de cenas: o processo de criação do grafo de cena envolve a colocação aleatória de objectos na cena e a definição de parâmetros introduzidos pelo utilizador. Estas duas características produzem uma estrutura de dados hierárquica única. Desta forma, torna-se necessário guardar o grafo de cena resultante em ficheiro para permitir o seu uso repetidas vezes. As cenas são guardadas no formato próprio do OSG (*.osg*).

- Inserção automática de nós OQ: um dos objectivos da aplicação desenvolvida é permitir estudar o efeito do uso dos testes de oclusão por *hardware* em diferentes níveis hierárquicos, e combinações destes. Assim, foi adicionada a funcionalidade que permite inserir nós OQ nos níveis hierárquicos do grafo de cena seleccionados pelo utilizador. Desta forma, em diferentes simulações, a mesma cena pode ser testada com diversas configurações de nós OQ.

- Gestão de percursos: para o estudo mencionado no ponto anterior, são realizadas simulações onde é percorrido um trajecto na cena 3D. A gravação de um percurso consiste na escrita para ficheiro da posição e orientação da câmara enquanto esta avança a uma velocidade constante. Os dados são recolhidos com uma frequência de 25 Hz e enviados para um ficheiro de texto. Durante a simulação os valores da posição e orientação são lidos e usados por um manipulador da câmara desenvolvido especificamente para este efeito.

- Recolha de estatísticas: como mencionado anteriormente, o OSG permite a apresentação de diversas estatísticas da simulação no ecrã. No entanto, medidas específicas para analisar o desempenho, no caso particular da remoção hierárquica por oclusão, não são recolhidas pela implementação da API. Os valores recolhidos são guardados em ficheiro para posterior análise. Exemplos de estatísticas recolhidas, adicionais às do OSG, são a duração de cálculo de cada *frame*, o número de testes de oclusão por *hardware* realizados por *frame*, o número de nós OQ por *frame* considerados visíveis, oclusos e removidos do *view-frustum*.

Estatísticas que envolvem percorrer toda a estrutura hierárquica constituem um custo de processamento não desprezável. Este varrimento do grafo de cena, a ser realizado todas as *frames* em estruturas com um elevado número de nós a visitar, aumenta o tempo do cálculo da *frame*. Desta forma, nas simulações realizadas não se recolheram todos os dados possíveis, apenas os necessários para o estudo em causa.

6.2.2. ALTERAÇÕES À API DO OSG

Para concretizar algumas das funcionalidades descritas na secção anterior e a abordagem alternativa ao algoritmo de remoção por oclusão original do OSG, apresentada na secção seguinte, revelou-se necessário modificar a API. Estas alterações incidem maioritariamente nas classes **osg::OcclusionQueryNode** e **osgUtil::CullVisitor**. As três modificações mais relevantes são apresentadas e justificadas.

- Como descrito na Secção 5.2 o algoritmo de remoção por oclusão do OSG usa um intervalo fixo entre testes de oclusão por *hardware* definido em *frames*. Esta primeira alteração à API consiste na definição do intervalo entre testes de oclusão por *hardware* como um período de tempo. Uma simulação realizada a *frame rate* constante pode não conseguir atingir a cadência de imagens pré-definida. Neste caso, a aplicação vai dispor de menos informação para determinar a visibilidade correctamente se o intervalo entre testes de oclusão por *hardware* for definido em *frames*. O decréscimo nas *frames* por segundo implica uma diminuição do número de testes de oclusão por *hardware*, nesta situação uma maior quantidade de geometria é erradamente ou desnecessariamente desenhada o que afecta o desempenho. A mesma situação ocorre em simulações realizadas sem limite de *frames* por segundo (*free run*), onde a flutuação do número de imagens geradas por segundo irá também afectar o número de testes de oclusão por *hardware* realizados.

- A segunda alteração à API consiste em detectar quando um nó OQ reentra no *view-frustum.* Este procedimento permite testar e reiniciar o contador de tempo/*frames* do nó OQ desde o último teste de oclusão por *hardware*. Na versão actual, a um nó OQ que transite para o *view-frustum*, o teste de oclusão por

hardware só lhe é aplicado depois de excedido o intervalo de tempo/*frames* iniciado antes de sair do *view-frustum*. Por outras palavras, um nó pode sair e entrar no *view-frustum* em diferentes situações de visibilidade sem que lhe seja aplicado um teste de oclusão por *hardware*, desde que esta situação ocorra dentro do intervalo pré-definido entre testes. Este funcionamento tem particular relevância quando é aplicada a coerência temporal a nós oclusos e a frequência de entrada/saída de nós OQ no *view-frustum* é elevada.

- A terceira alteração consiste em permitir a definição de um intervalo entre testes de oclusão por *hardware* diferente para nós oclusos e nós visíveis. Actualmente, o mesmo intervalo de *frames* é usado nos dois casos. Assim, ao definir intervalos diferentes é possível afinar o impacto que a aplicação da coerência temporal tem na aplicação, quando aplicada a nós OQ com diferentes estados de visibilidade.

6.2.3. ALGORITMO DE REMOÇÃO POR OCLUSÃO ALTERNATIVO AO OSG

Uma implicação directa da duração do intervalo de tempo entre testes de oclusão por *hardware* é a quantidade de resultados disponíveis para determinar a visibilidade. O uso de um intervalo entre testes visa diminuir a influência negativa que estes têm no desempenho da aplicação. No entanto, a ausência de informação para determinar a visibilidade correctamente em cada *frame* pode ter um efeito mais penalizador que a realização de testes com maior frequência, como abordado na Secção 5.2.1.

O OSG, ao aplicar a coerência temporal a nós anteriormente oclusos está a assumir que estes tendem a permanecer oclusos durante um período de tempo. Se durante este período a geometria se torna visível, como ainda é considera oclusa, não é desenhada. Esta situação tem consequências directas na correcção da imagem gerada ao não apresentar ao utilizador objectos que na realidade se encontram visíveis. Esta perda de objectos visíveis implica que esta geometria não será usada como oclusora. Neste caso, será desenhada geometria que poderia ser identificada como oclusa caso todos os objectos realmente visíveis fossem desenhados. A somar ao custo de desenhar esta geometria, há o custo da realização de testes de oclusão por *hardware* associados a esta, também desnecessários. Este problema é endereçado pelos algoritmos de remoção por

oclusão descritos no Capítulo 4, aplicando em todas as *frames* o teste de oclusão por *hardware* a geometria determinada como oclusa na *frame* anterior.

Assim, o algoritmo de remoção do OSG foi modificado para realizar testes de oclusão por *hardware* a nós oclusos em todas as *frames*, o que implica necessariamente um aumento no número de testes por *frame* relativamente ao algoritmo original. Enquanto a aplicação da coerência temporal a nós oclusos tem impacto na quantidade de geometria erradamente desenhada, quando aplicada a nós visíveis a consequência é o aumento da geometria desnecessariamente desenhada. Ao continuar a ser considerada como visível, a geometria é enviada para o *hardware* gráfico numa altura em que já pode estar oclusa. Esta situação não afecta a correcção da imagem, apenas aumenta a quantidade de geometria a processar o que se reflecte no tempo de cálculo de cada *frame*. Assim, para melhorar o desempenho da aplicação e gerar uma imagem correcta, o número de testes aos nós visíveis deve ser optimizado.

Uma abordagem ao problema do teste a nós visíveis foi proposta por Kovalcik e Sochor em [KOV05]. Os autores propõem que o intervalo entre testes de oclusão por *hardware* a um nó visível seja determinado com base no número de vezes que um nó foi determinado visível. De uma forma simples, se um nó é sucessivamente calculado como visível, então a probabilidade de assim continuar é elevada. Desta forma, a aplicação de testes ao nó visível pode ser feita mais espaçadamente com o avançar do tempo o que significa menos despesa na realização de testes de oclusão por *hardware*. Esta ideia foi mais tarde incorporada por Mattausch *et al.* no algoritmo CHC++ [MAT08].

As alterações introduzidas no algoritmo do OSG para realizar testes de oclusão por *hardware* com um intervalo dependente do historial do nó visível são apresentadas na Figura 33. O intervalo de tempo entre testes é calculado pela função:

$$novo\_intervalo\_de\_tempo(i) = TEMPO\_BASE * (0{,}99 - 0{,}7e^{-i})$$

O valor do *TEMPO_BASE* define a gama de intervalos de tempos a usar, este valor deverá ser ajustado de acordo com o tipo de cena e velocidade do ponto de vista. O parâmetro *i*, representa o número de vezes que um nó visível é testado e retorna o mesmo estado de visibilidade. Este parâmetro é reiniciado sempre que o nó é determinado como ocluso.

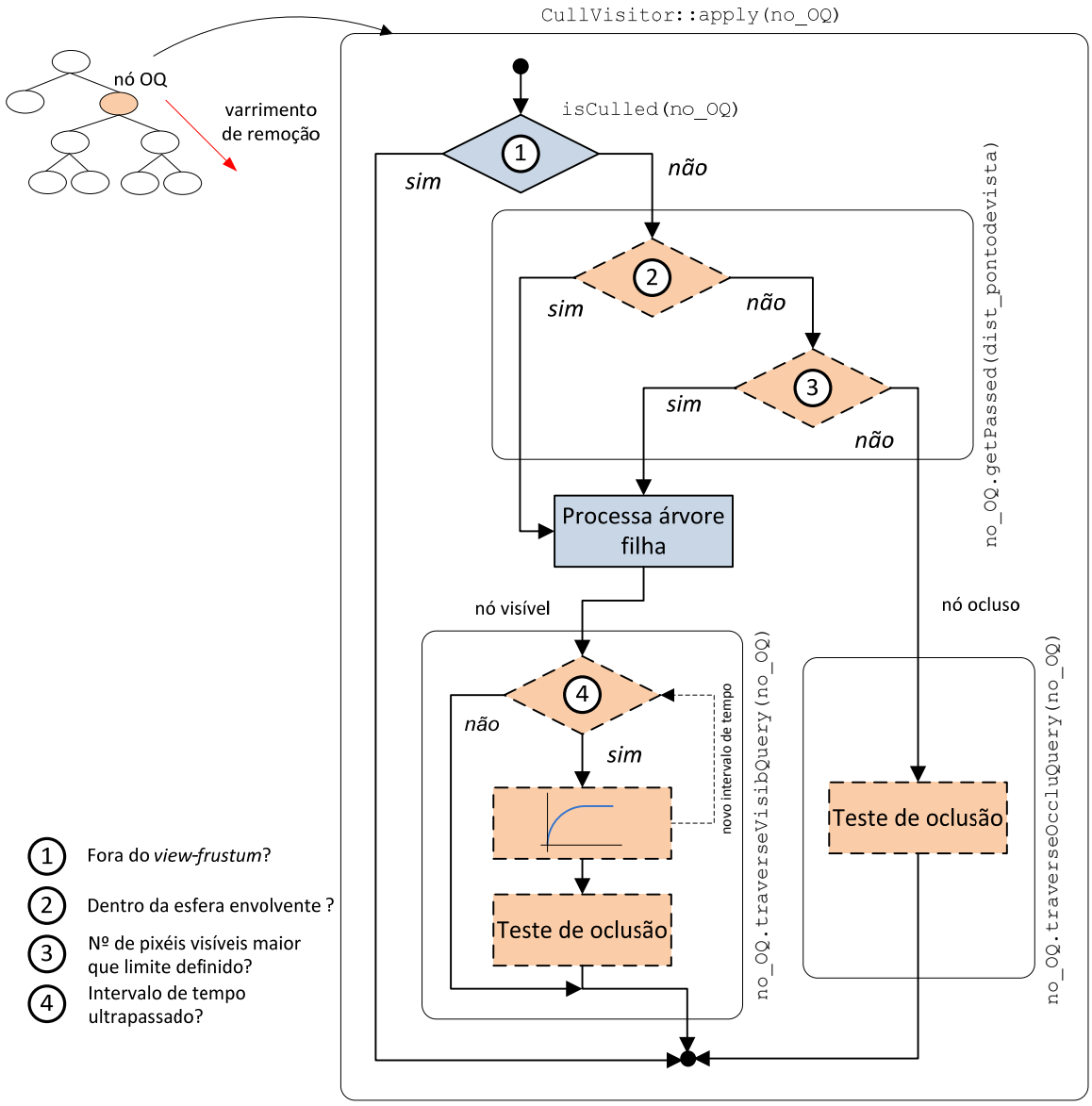


Figura 33 – Algoritmo de remoção por oclusão do OSG modificado.

A função anterior foi adaptada da descrita no algoritmo CHC++, explicada em detalhe na Secção 4.5. A função do CHC++ foi obtida como uma aproximação a valores recolhidos em simulação onde foi estudado o historial da visibilidade dos nós da cena. A Figura 34 ilustra um exemplo onde a curva que permite obter os intervalos entre testes de oclusão por *hardware* a nós visíveis foi obtida pela função *novo_intervalo_de_tempo* para um *TEMPO_BASE* de 240 ms. Neste exemplo, o intervalo inicial entre testes de oclusão por *hardware* a nós visíveis começa em 70 ms e evolui até aos 238 ms à medida que o nó em causa se mantenha visível.

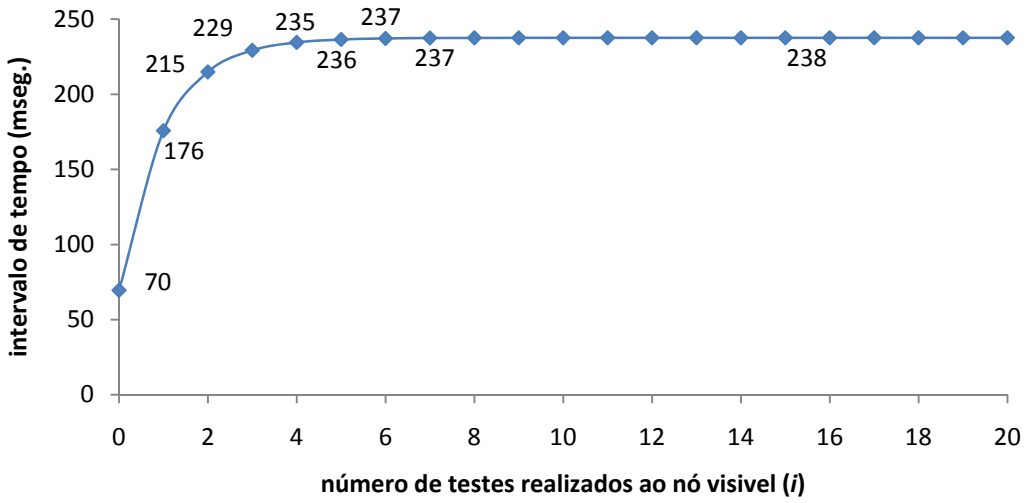


Figura 34 - Intervalos entre testes a nós visíveis para um *TEMPO_BASE* de 240 ms.

6.3. CENAS USADAS NAS SIMULAÇÕES

Para testar como a realização de testes de oclusão por *hardware* em vários níveis hierárquicos da descrição de uma cena afecta o desempenho da aplicação, foram usadas duas cenas estáticas de muito grandes dimensões com diferentes níveis de densidade de geometria. As cenas criadas são do tipo ambiente urbano onde os objectos são distribuídos numa superfície essencialmente plana mas, neste caso, não se encontram alinhados. Este tipo de distribuição proporciona um poder oclusor com maior diversidade ao contrário de uma típica cena urbana onde os edifícios se encontram alinhados com as ruas.

De uma forma genérica, todas as cenas foram criadas distribuindo aleatoriamente objectos num plano. Estes objectos constituem os nós folha. A estrutura hierárquica compreende 4 níveis: nó raiz, dois níveis de nós grupo e nós folha. Para atribuir nós folha aos nós grupo imediatamente acima na hierarquia são inseridos objectos, em número configurável, em áreas quadradas no plano. Estas áreas dividem o plano uniformemente e são de dimensão e em quantidade configuráveis. O nível hierárquico de nós grupo, pais dos nós folha, é por sua vez agrupado por um número também configurável de nós grupo, os quais são todos filhos do nó raiz.

Os parâmetros usados na criação do grafo de cena foram escolhidos para que o grafo seja equilibrado. Neste caso, em cada cena foram usadas 1024 áreas, *i.e.*, nós grupo acima dos

nós folha. Estes nós grupo são, por sua vez, agrupados em conjuntos de 4x4 pelos nós grupo do nível acima. A quantidade de áreas usada ocupa uma superfície quadrada de 8,6 kms de lado.

As cenas são constituídas por dois tipos de objectos: edifícios e pequenos objectos. A existência de edifícios proporciona boa oclusão mas simultaneamente são estes os objectos que interessa remover visto serem os que representam um maior custo na geração da imagem final pela sua quantidade e contribuição em termos de área a desenhar. Os pequenos objectos usados não são bons oclusores. Encontram-se dispersos e são de pequena complexidade. Nas simulações descritas na secção seguinte, os pequenos objectos não são sujeitos ao teste de oclusão por *hardware* quando realizado ao nível da geometria. O custo de processamento destes é baixo quando comparado com o dos edifícios. Caso lhes fosse aplicado o teste de forma individual isso adicionaria a cada *frame* o custo dos testes sem que o benefício, quando oclusos, fosse significativo. A Figura 35 apresenta o aspecto geral das cenas criadas: **Cena 1** e **Cena 2**.

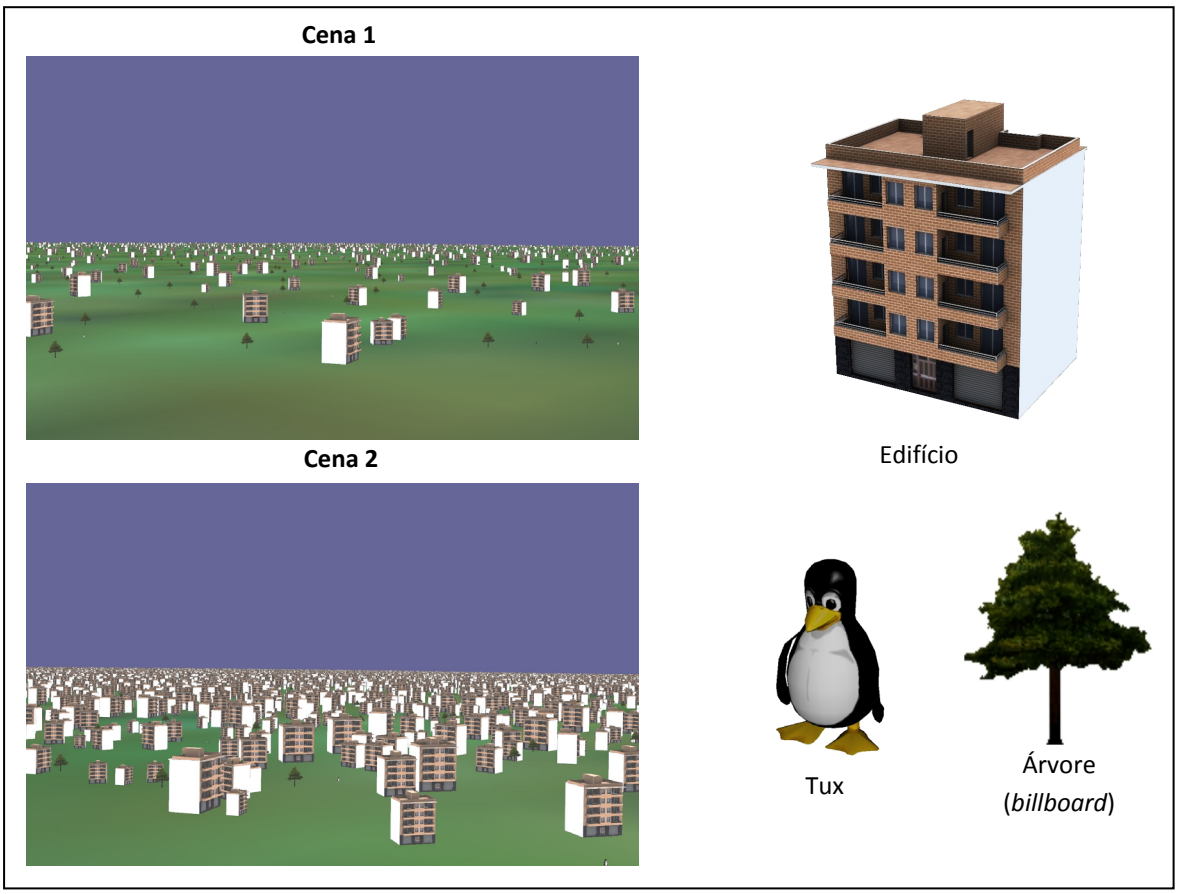


Figura 35 – As duas cenas criadas para as simulações e os objectos usados para as povoar.

As duas cenas usadas nas simulações têm a mesma dimensão mas níveis de complexidade geométrica diferentes, devido à variação da quantidade de objectos inseridos em cada região. Algumas estatísticas das cenas são apresentadas na Tabela 2. Os objectos usados na **Cena 1** ocupam aproximadamente 2% da área do plano enquanto os da **Cena 2** ocupam 22%, o que proporciona um bom poder oclusor. Para adicionar alguma diversidade aos oclusores, os edifícios foram rodados e escalados aleatoriamente durante o processo de inserção.

Tabela 2 - Algumas estatísticas das cenas usadas nas simulações.

	#edifícios	#outros	#vértices	#primitivas
Cena 1	2 028	4 056	14 920 409	5 165 020
Cena 2	25 426	10 862	160 950 781	45 527 127

6.4. SIMULAÇÕES

A realização de testes de oclusão por *hardware* numa aplicação que usa a API do OSG requer o uso de nós OQ. Para analisar como a realização de testes de oclusão por *hardware* em vários níveis hierárquicos influência o desempenho da aplicação, em cada simulação, foram introduzidos nas cenas criadas nós OQ com diferentes configurações. Uma configuração representa o conjunto de níveis hierárquicos ao qual é aplicado o algoritmo de remoção por oclusão. Cada simulação consiste em usar uma das configurações, para cada uma das duas cenas. A Figura 36 ilustra a relação entre as configurações de nós OQ e a estrutura hierárquica das cenas criadas:

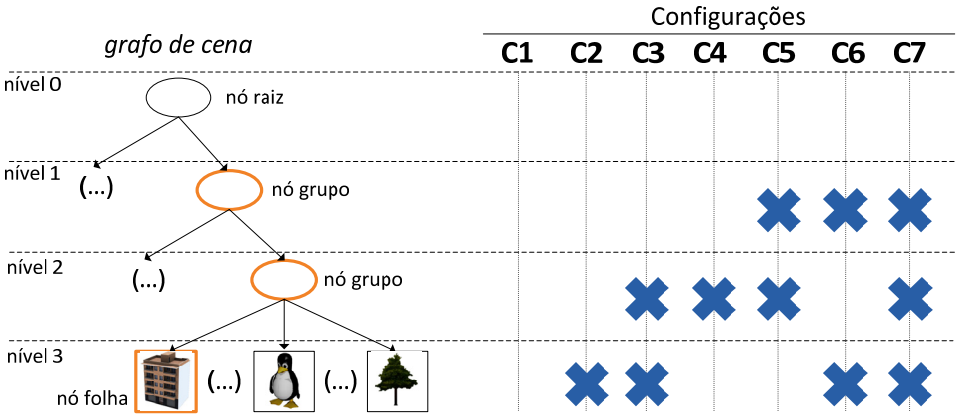


Figura 36 - Configurações de nós OQ usadas nas simulações de cada cena.

O processo de inserção de nós OQ realiza-se substituindo nos níveis intermédios da hierarquia os nós grupo por nós OQ e ao nível da geometria adicionar nós OQ como pais dos nós folha que contêm a geometria.

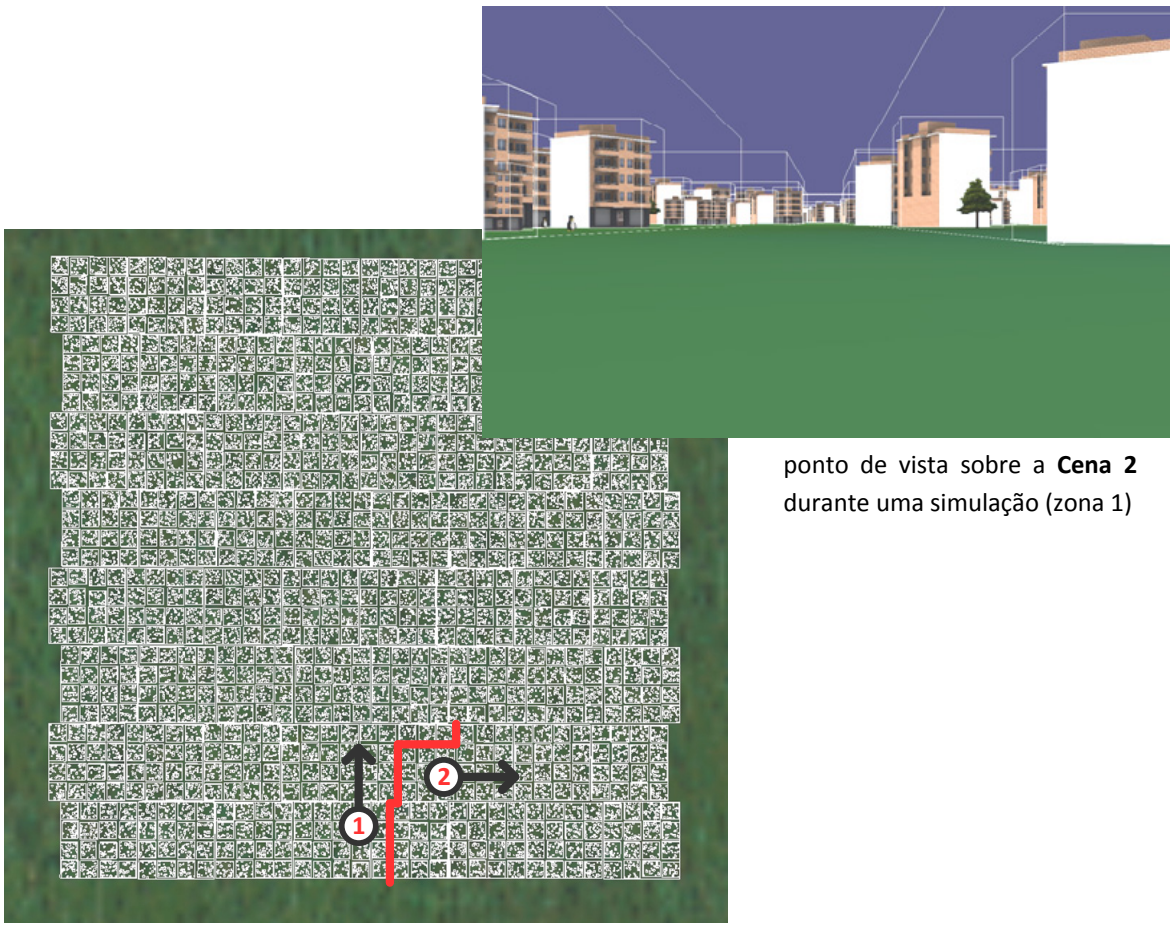
- Configuração **C1**: com esta configuração não são inseridos nós OQ, apenas é realizado o *view-frustum culling* para estabelecer uma base de comparação;
- Configuração **C2**: são inseridos nós OQ apenas no nível hierárquico 3, como pais dos nós folha que contém a geometria do modelo do edifício;
- Configuração **C3**: para além do nível 3 também são inseridos nós OQ no nível 2, como nós que agrupam os nós folha;
- Configuração **C4**: apenas são inseridos nós OQ no nível hierárquico 2;
- Configuração **C5**: são inseridos nós OQ no nível 2 e nível 1;
- Configuração **C6**: são inseridos nós OQ no nível 1 e ao nível da geometria (nível 3);
- Configuração **C7**: são inseridos nós OQ em todos os níveis hierárquicos com excepção do nó raiz.

Do conjunto de configurações de nós OQ possível não é usada a configuração que realizaria testes de oclusão por *hardware* apenas no nível hierárquico 1. Tal como enunciado na Secção 6.1, a realização de testes apenas no nível hierárquico superior implicaria que muita geometria fosse enviada para o *hardware* gráfico mesmo que oclusa. Devido à dimensão dos volumes envolventes associados a este nível hierárquico, a probabilidade de estes serem determinados como oclusos é reduzida. Nesta situação, a ausência de um meio para resolver a visibilidade nos níveis inferiores da hierarquia faria com que o desempenho se aproximasse do *view-frustum culling*.

A principal métrica adoptada para avaliar o desempenho da aplicação é a duração de cálculo de cada *frame* ao longo da simulação, realizada a *frame rate* constante. Como descrito no Capítulo 5, os nós OQ permitem a configuração de dois parâmetros. Um dos quais é o intervalo entre testes de oclusão por *hardware* definido em número de *frames*. No entanto, em situações pontuais onde a aplicação não consegue atingir a *frame rate*

proposta o número de testes realizados decai o que afecta a determinação de visibilidade. Este problema foi contornado com a alteração para um intervalo de tempo entre testes de oclusão por *hardware*, descrita na Secção 6.2.2. O valor usado foi de 0,167 segundos o qual representa um período equivalente a um intervalo de 5 *frames* (valor padrão usado pelo OSG) numa simulação realizada a 30 fps constantes. O restante parâmetro configurável, o número de píxeis a partir do qual um volume envolvente testado é considerado visível, foi definido de forma conservadora com o valor de 1 píxel.

A recolha de dados das simulações decorre durante um percurso previamente gravado onde a câmara percorre a cena na forma de *walkthrough*. Isto significa que a câmara deslocasse-se essencialmente paralela e próximo à base da cena. O trajecto é percorrido a velocidade constante e leva o ponto de vista sobre a cena a visitar zonas com diferentes complexidades em profundidade e densidades de geometria. O trajecto é percorrido em 106 segundos e a sua localização na cena é ilustrada na imagem à esquerda na Figura 37.



ponto de vista sobre a **Cena 2** durante uma simulação (zona 1)

Figura 37 – Duas perspectivas do percurso usado nas simulações.

A imagem à esquerda na Figura 37, apresenta o percurso (a vermelho) onde os pontos 1 e 2 indicam duas zonas de diferente complexidade em profundidade. A Figura 37 também ilustra, na imagem à direita, um ponto de vista sobre a **Cena 2** no inicio do trajecto onde é possível verificar a oclusão que a densidade de geometria usada nesta cena permite. Nesta imagem, a visualização dos volumes envolventes encontra-se activa. Durante uma simulação diversos dados são recolhidos uma vez por *frame* e guardados em disco para posterior análise do desempenho.

Os resultados das simulações a apresentar na secção seguinte foram obtidos usando o algoritmo original de remoção por oclusão implementado pelo OSG, apenas com a alteração relativa ao intervalo entre testes ser definido em tempo. Este algoritmo decide a geometria a remover com base no resultado do teste de oclusão por *hardware* realizado pelo nó OQ e realiza novo teste a todos os nós desde que tenha decorrido um intervalo, neste caso de tempo, desde o último teste a esse nó. Na Secção 6.6 serão apresentados resultados de simulações relativos às alterações propostas ao algoritmo de remoção por oclusão original do OSG, descritas na Secção 6.2.3.

Por fim, todas as simulações realizadas no âmbito desta dissertação foram conduzidas num computador pessoal equipado com um CPU Intel Core2 Duo a 2,53 GHz, 3 GB de memória RAM, uma placa aceleradora gráfica NVidia GeForce 9650m GT com 1 GB de memória e com o sistema operativo Windows Vista Home Premium (32 bits) instalado.

6.5. ANÁLISE DE RESULTADOS

Nesta secção são apresentados os resultados das simulações realizadas para estudar a influência que a complexidade da cena e a aplicação hierárquica dos testes de oclusão por *hardware* têm no desempenho da aplicação 3D usando a abordagem de remoção de geometria proposta pelo projecto OSG, anteriormente descrita.

6.5.1. CENA 1

A **Cena 1**, usada para este conjunto inicial de simulações, é de grandes dimensões mas de baixa complexidade geométrica. Os edifícios encontram-se dispersos o que não favorece a probabilidade de oclusão, incluindo a fusão de oclusores, e leva a que objectos distantes

sejam facilmente visíveis. Nesta situação, a aplicação de testes de oclusão por *hardware* a níveis hierárquicos que envolvam geometria dispersa por uma grande área não se revela proveitoso já que os volumes envolventes são dificilmente determinados como oclusos.

A Figura 38 apresenta o resultado de quatro simulações onde cada curva representa a aplicação do algoritmo de remoção por oclusão do OSG relativamente às configurações **C1**, **C2**, **C3** e **C4** descritas na Figura 36. Com a configuração **C1** apenas o *view-frustum culling* é usado para remover geometria.

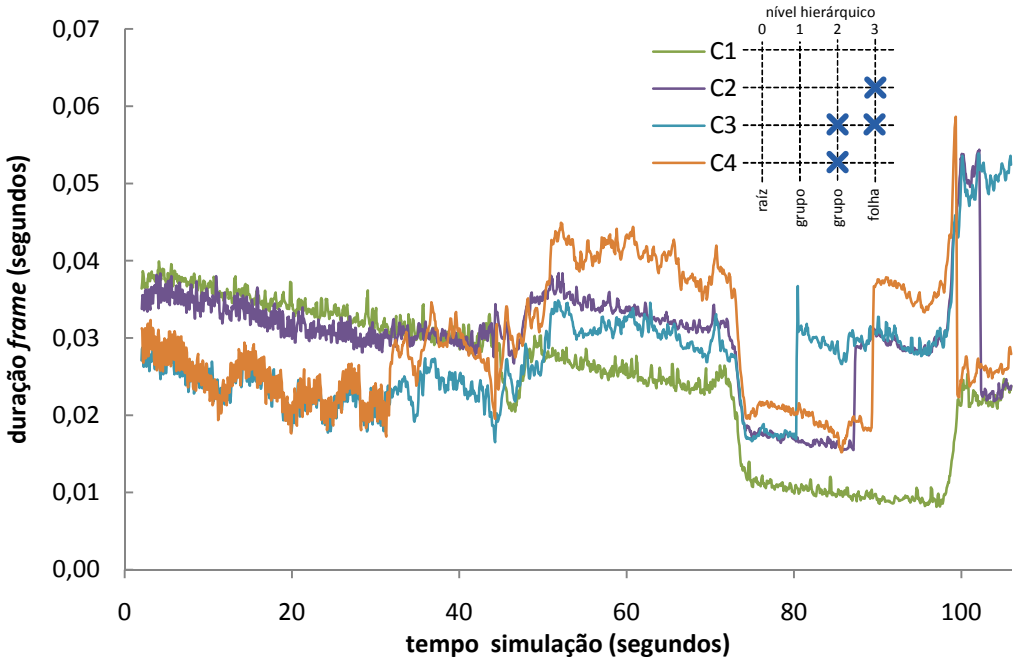


Figura 38 – Resultados das simulações com as configurações C1, C2, C3 e C4 aplicadas à Cena 1.

Os primeiros 50 segundos das simulações decorrem numa zona da cena onde a densidade de geometria é suficiente para que a duração de cálculo de uma *frame* beneficie com a aplicação dos testes de oclusão por *hardware*. Apesar de pouco significativa, a duração de cada *frame* é inferior à configuração C1 onde apenas é aplicada a técnica de remoção de geometria que se encontra fora do *view-frustum*.

Durante este período inicial, as configurações que envolvem a aplicação dos testes de oclusão por *hardware* aos nós grupo no nível 2 da hierarquia (**C3** e **C4**), apresentam o melhor desempenho devido a permitirem a remoção de significativas quantidades de objectos com um único teste.

Na restante duração das simulações, a aplicação apenas do *view-frustum culling* (**C1**) apresenta melhor desempenho que qualquer outra configuração. Neste caso, a realização de testes de oclusão por *hardware* resulta na classificação maioritária da geometria como visível, devido à baixa probabilidade de oclusão. O custo de realizar os testes e desenhar a geometria não é compensado pelo ganho de não desenhar a pouca geometria determinada como oclusa. No início do percurso, as configurações **C3** e **C4** apresentam um comportamento semelhante, mas com o avançar das simulações verifica-se uma divergência no impacto que estas configurações têm no desempenho da aplicação. Com a configuração **C4**, os testes apenas são aplicados ao nível 2 da hierarquia. Assim, quando a geometria começa a ficar mais dispersa não existe forma de remover individualmente objectos que se encontrem oclusos, ao contrário do que acontece nas configurações **C2** e **C3**.

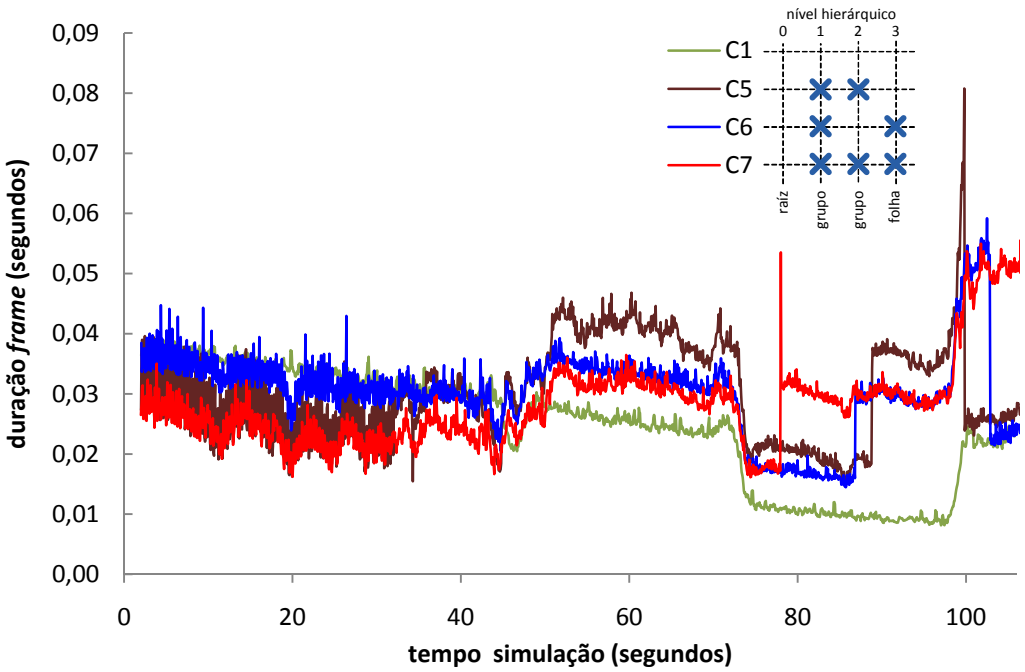


Figura 39 – Resultados das simulações com as configurações C5, C6 e C7 aplicadas à Cena 1.

A Figura 39 apresenta os resultados das restantes configurações quando aplicadas à **Cena 1**. As configurações **C5**, **C6** e **C7** destacam-se por realizar os testes de oclusão por *hardware* aos nós grupos situados no nível 1 da hierarquia o que significa que os volumes envolventes associados a estes nós são de muito maior dimensão. Nas duas cenas criadas para este estudo, os nós grupo do nível 1 englobam 16 nós grupo do nível 2.

Na globalidade verifica-se o mesmo comportamento que nas simulações analisadas anteriormente. Nos primeiros 50 segundos as configurações que usam testes de oclusão por *hardware* originam um melhor desempenho que o *view-frustum culling*, o que após este período inicial deixa de se verificar. Das várias configurações destaca-se a **C6** que aplica testes de oclusão por *hardware* ao nível 1 e 3 da hierarquia. Esta configuração salienta a importância do nível 2 quando comparada com a configuração **C7** nas zonas de maior densidade de geometria. Ao mesmo tempo, comprova que a grande dimensão dos volumes envolventes do nível 1 e a sua baixa probabilidade de serem determinados como oclusos, pouco impacto tem na aplicação ao ter um comportamento idêntico à configuração **C2**.

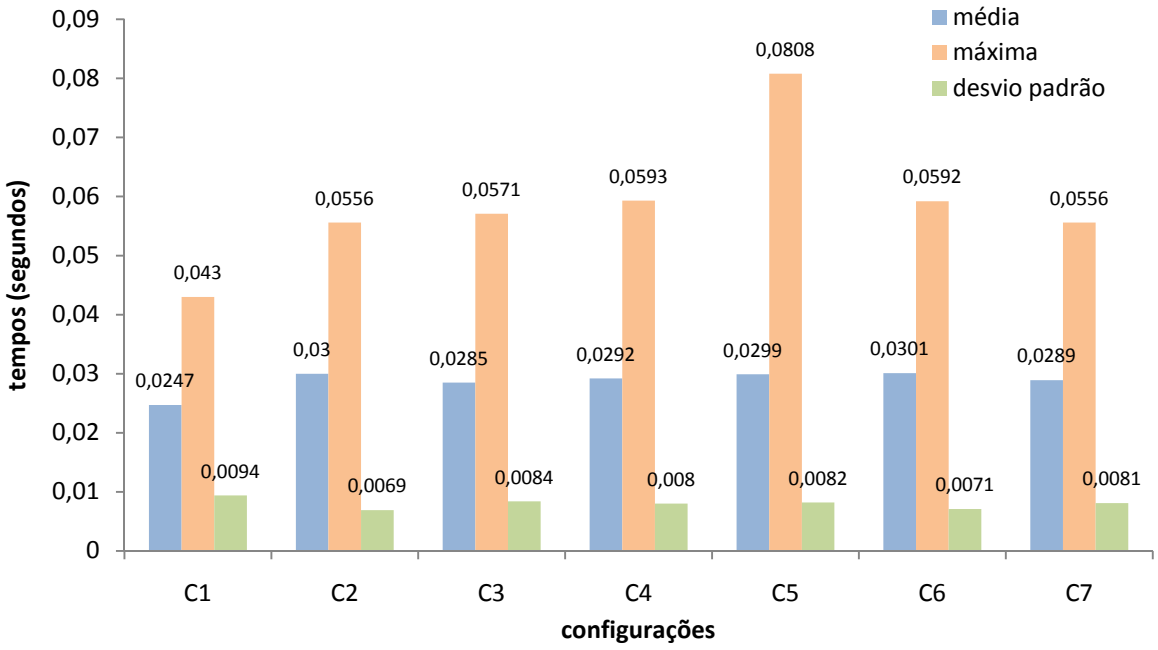


Figura 40 – Cena 1: duração média, valor máximo e desvio padrão das *frames* nas simulações.

Com base nos valores apresentados na Figura 40, a configuração **C1** proporciona globalmente o melhor desempenho ao apresentar o menor valor da duração média de cálculo de *frames*. No entanto, das configurações que usam testes de oclusão por *hardware*, a **C3** e a **C7** apresentam desempenhos próximos o que salienta que a existência da possibilidade de remover grupos de geometria com um único teste favorece a remoção hierárquica por oclusão.

6.5.2. CENA 2

Os resultados das simulações anteriores evidenciam tendências, com resultados promissores em zonas da cena de maior densidade de geometria, mas não são conclusivos quanto às vantagens de realizar testes de oclusão por *hardware* de uma forma generalizada. A **Cena 2** tema a mesma dimensão que a **Cena 1** mas com doze vezes mais edifícios na mesma área, possui uma densidade de geometria bastante superior, o que favorece a oclusão.

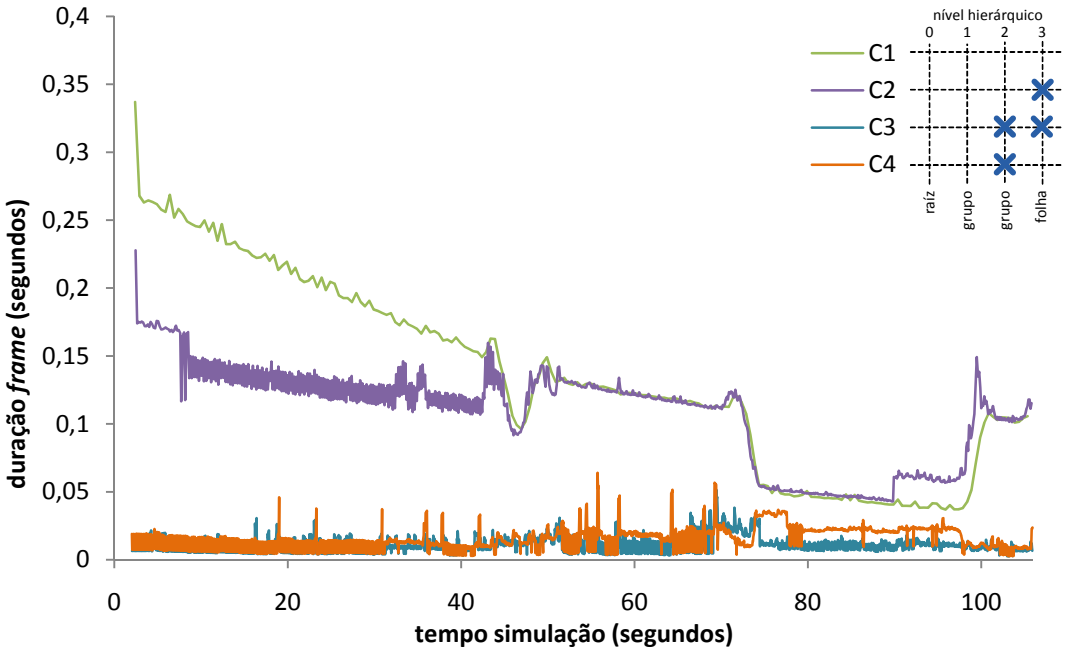


Figura 41 - Cena 2: resultados das simulações realizadas com as configurações C1 a C4.

Do conjunto de resultados apresentados na Figura 41, relativos a várias simulações com a **Cena 2**, observa-se:

- A aplicação não hierárquica dos testes de oclusão por *hardware* (**C2**) apresenta um ganho modesto face a **C1**. Na zona inicial do percurso de maior densidade consegue remover por oclusão alguma geometria, mas à medida que a complexidade diminui o desempenho aproxima-se do puro *view-frustum culling*. A elevada quantidade de testes de oclusão por *hardware* necessários para determinar a visibilidade reflecte-se no desempenho.

- O uso das configurações **C3** e **C4** revela uma significativa e constante diminuição no tempo de cálculo das *frames*. O ponto em comum destas duas configurações é a aplicação dos testes de oclusão por *hardware* ao nível hierárquico 2 (superior à geometria), neste caso, com a realização de um único teste é possível remover todos os nós de geometria abrangidos por um nó grupo. A densidade de geometria usada nesta cena permite que existam permanentemente bastantes grupos oclusos o que se reflecte no desempenho ao evitar desenhar todos os objectos de um grupo com um único teste de oclusão por *hardware*.

- A principal diferença entre as configurações **C3** e **C4** localiza-se na zona de menor complexidade, a partir dos 72 segundos sensivelmente. Enquanto ambas as configurações conseguem remover grupos inteiramente oclusos, com **C3** é possível remover por oclusão nós de geometria numa situação onde C4 terá que desenhar todos esses nós. Nas zonas de menor densidade menos grupos são determinados como oclusos, com **C3** o estado de visibilidade é refinado ao testar individualmente os edifícios enquanto com **C4**, se um grupo for determinado como visível todos os objectos por si abrangidos são desenhados o que se traduz em mais tempo de processamento em cada *frame*.

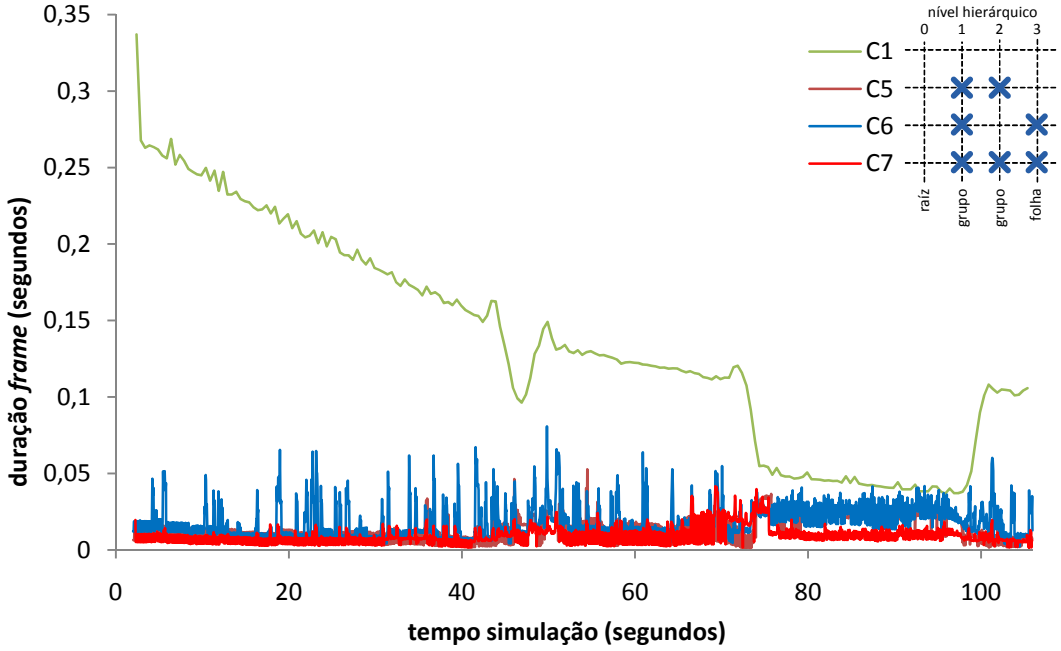


Figura 42 - Cena 2: resultados das simulações realizadas com as configurações C5, C6 e C7.

Além das quatro configurações apresentadas na Figura 41, são apresentados na Figura 42 os resultados das simulações com as restantes três configurações aplicadas à **Cena 2**. A resposta com a configuração **C1** é mantida para comparação. A linha referente à configuração **C5**, pouco visível na figura, apresenta um comportamento semelhante à **C7** até os 72 segundos, sensivelmente. Na zona de menor complexidade, é coincidente com a linha da configuração **C6**.

As configurações **C5**, **C6** e **C7** representam a aplicação dos testes de oclusão por *hardware* ao nível hierárquico 1. A inclusão deste nível nos testes não proporciona ganhos relativamente a **C3** e **C4** na mesma ordem de grandeza que estes evidenciam relativamente a **C1**. Apesar disso, demonstram uma melhoria no desempenho proporcionada pela remoção, mesmo que momentânea, de elevadas quantidades de objectos. Como verificado nas simulações com a **Cena 1**, este comportamento só é possível se a oclusão na cena for significativa tendo em consideração a dimensão dos volumes envolventes testados no nível hierárquico 1.

Na Figura 43 é apresentado o gráfico comparativo de diversos valores numéricos obtidos das simulações realizadas para a **Cena 2** com as sete configurações estudadas.

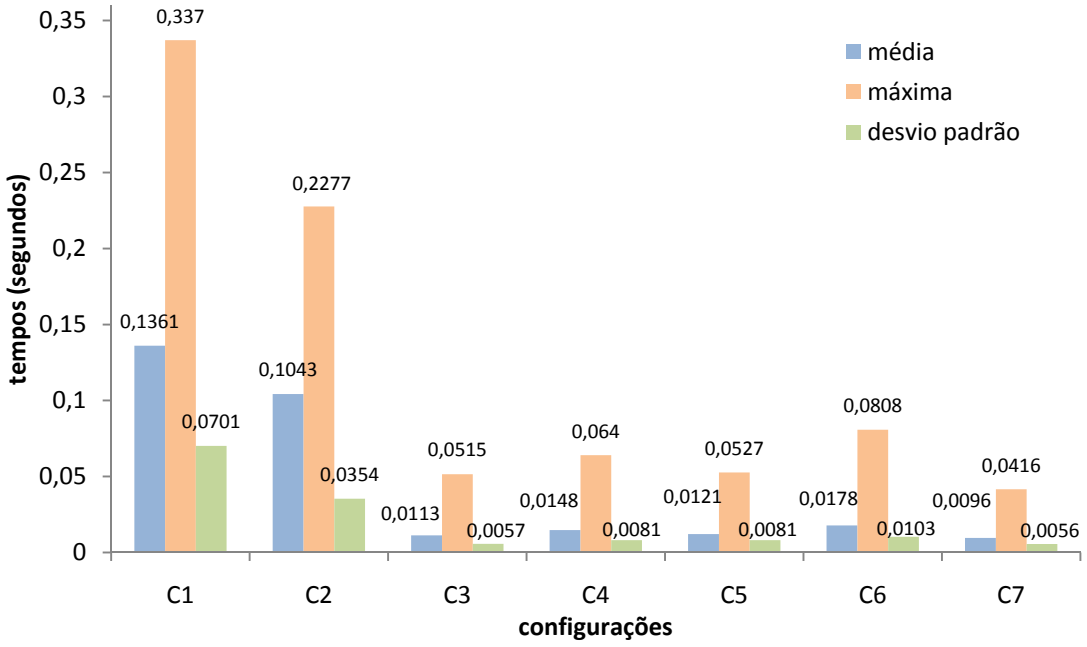


Figura 43 - Cena 2: duração média, valor máximo e desvio padrão das *frames* nas simulações.

Os valores numéricos apresentados na Figura 43 mostram que, relativamente ao conjunto de simulações da Figura 42, as configurações **C5** e **C7** apresentam os melhores tempos médios de cálculo de *frames*. Ambas as configurações aplicam os testes ao nível hierárquico 2. Com a configuração **C6**, o nível 2 não é usado nos testes de oclusão por *hardware* o que se reflecte na duração média das *frames*. Neste caso, o tempo por *frame* sobe para valores superiores a qualquer uma das configurações (**C3**, **C4**, **C5** e **C7**) que usam os testes no nível hierárquico 2, acima da geometria. Como mencionado na Secção 6.1, a realização de testes em níveis hierárquicos não adjacentes implica uma grande oscilação na performance verificável na Figura 43 e pelo valor do desvio padrão apresentado pela configuração **C6**.

Como resultado da aplicação dos testes em todos os níveis hierárquicos e beneficiando das vantagens de cada um, a configuração **C7** apresenta o melhor e mais equilibrado desempenho durante a simulação com a **Cena 2**. O desempenho de todas as configurações que usam testes no nível 2, quando comparado com as configurações **C1**, **C2** e **C6**, demonstram que este é o nível hierárquico mais importante a testar. Esta melhoria no desempenho é explicada pela capacidade que estes nós grupo têm em remover bastantes objectos em conjunto com a sua relativa pequena dimensão o que permite que seja determinados oclusos com facilidade. O impacto de testar o nível hierárquico 1 será mais significativo em cenas de muito maior dimensão que a **Cena 2**, pelas mesmas razões.

Outra forma de analisar a influência das diferentes configurações na aplicação 3D é a quantidade de geometria efectivamente desenhada em cada *frame*. A Figura 44 ilustra essa diferença em vértices desenhados, em média, por *frame*.

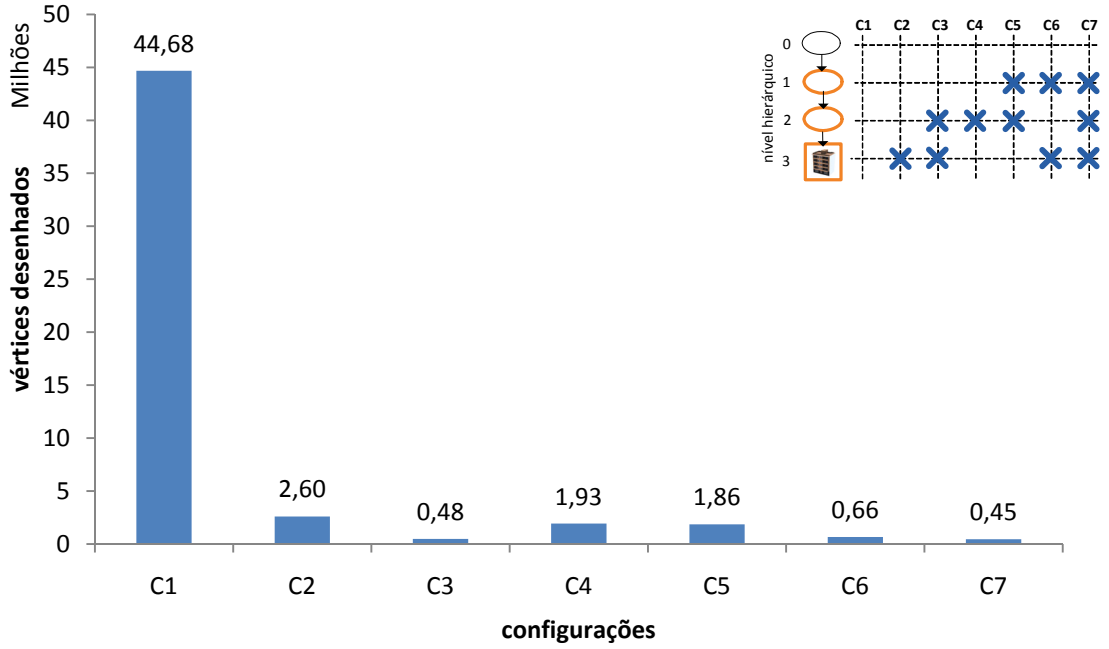


Figura 44 - Cena 2: quantidade média de vértices desenhados por *frame* em cada uma das configurações.

Ao comparar o gráfico da Figura 44 com o da Figura 43, verifica-se que certas configurações que conseguem remover por oclusão elevadas quantidades de geometria não apresentam equivalentes reduções no tempo de processamento de cada *frame*. Este facto deve-se à adição do custo de processamento dos testes de oclusão por *hardware* ao tempo de cada *frame*. Esta situação é mais evidente quando são realizados grandes quantidades de testes, o que ocorre nas configurações **C2** e **C6**. Também se verifica que as configurações **C3**, **C6** e **C7** são as que apresentam os valores de vértices por *frame* mais reduzidos, o que indicia que para obter o conjunto potencialmente visível menos sobrestimado a combinação de níveis hierárquicos a usar nos testes de oclusão por *hardware* deve incluir o nível da geometria em conjunto com pelo menos um nível hierárquico superior que permita remover grupos de objectos.

6.5.3. Efeitos do Uso da Coerência Temporal

Um comportamento observável nas figuras que ilustram os resultados das simulações, são as oscilações de elevada frequência em algumas das respostas. O uso de um intervalo

fixo de tempo entre os testes a nós oclusos e nós visíveis propicia esta situação na medida em que a informação para determinar o estado de visibilidade não é continuamente actualizada. Nas *frames* que decorrem entre novos testes de oclusão por *hardware* o algoritmo baseia a sua decisão no último resultado existente. Este funcionamento significa que grandes quantidades de objectos podem transitar entre visíveis/oclusos no momento em que existem novos resultados do teste de oclusão por *hardware*.

A oscilação observável no tempo de cálculo das *frames* está directamente relacionada com a quantidade de objectos removidos/desenhados em conjunto com a contribuição dos testes de oclusão por *hardware* realizados aos nós filhos de um nó grupo considerado visível. Uma forma de minimizar esta situação seria, por exemplo, aumentar a frequência de testes aos nós OQ que no limite seria em todas as *frames*. Esta diminuição do intervalo de tempo adiciona mais custos de processamento mas ao mesmo tempo, ao conseguir determinar um conjunto potencialmente visível menos sobrestimado, também consegue remover mais geometria oclusa. O balanceamento entre estes factores depende da densidade de geometria da cena, velocidade de deslocação do ponto de vista e quantidade de geometria erradamente desenhada admissível na imagem final.

Outro factor que contribui para as oscilações das respostas, também resultante do uso de um intervalo fixo entre testes, é o alinhamento temporal de grandes quantidades de testes de oclusão por *hardware*. O uso de um intervalo fixo entre testes de oclusão por *hardware* em conjunto com a visibilidade simultânea de uma grande quantidade de nós sujeitos ao teste de oclusão por *hardware* provoca esta situação. Um exemplo desta situação é ilustrado pela Figura 45.

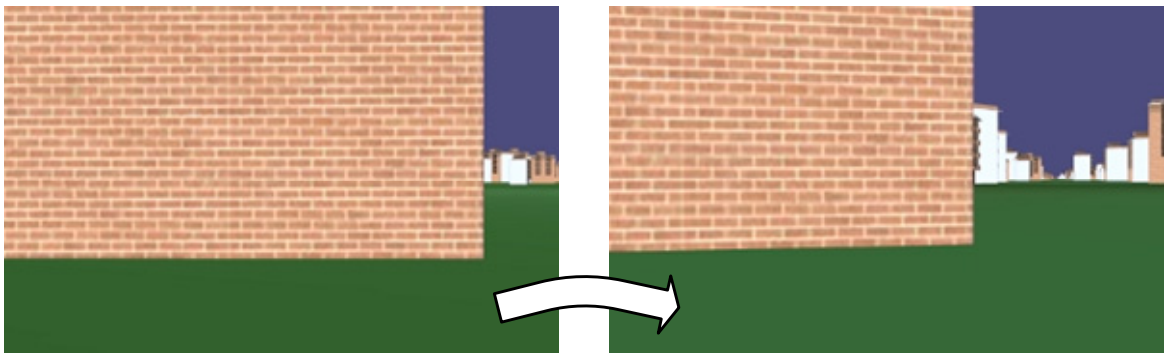


Figura 45 - Pequeno movimento do ponto de vista e o impacto na quantidade de geometria visível.

Todos estes nós, ao permanecerem no *view-frustum*, são sujeitos ao teste de oclusão por *hardware* repetidamente na mesma altura, o que propaga o efeito ao longo do tempo. Uma forma de reduzir este efeito é proposta pelo algoritmo CHC++ (Secção 4.5) pela distribuição aleatória dos testes de oclusão por *hardware* ao longo de um período de tempo em oposição à sua realização simultânea.

6.6. TESTES DE OCLUSÃO POR *HARDWARE* A NÓS VISÍVEIS

Todas as simulações apresentadas na Secção 6.5 foram realizadas com um intervalo fixo entre testes de oclusão por *hardware*, seguindo a abordagem original do OSG. Como anteriormente discutido (Secção 6.2.3), esta abordagem tem consequências directas na quantidade de geometria erradamente não desenhada. Em particular, quando a coerência temporal é aplicada a nós anteriormente oclusos onde a perda de objectos visíveis é mais evidente em movimentos rápidos da câmara e agravada por grandes intervalos entre testes. Neste sentido, o algoritmo de remoção por oclusão do OSG foi modificado para realizar testes de oclusão por *hardware* em todas as *frames* a nós considerados oclusos. Nesta fase, foi mantida a coerência temporal aplicada aos nós anteriormente visíveis e estudada a influência que o intervalo de tempo usado entre os testes a estes nós tem no desempenho da aplicação.

Para pequenos intervalos de tempo entre testes de oclusão por *hardware* a nós visíveis o desempenho degrada-se devido ao aumento da sua frequência. Ao aumentar o intervalo de tempo, atinge-se o pico de desempenho que representa a frequência ideal de testes de oclusão por *hardware*. No entanto, ao aumentar o intervalo, o desempenho piora devido à maior quantidade de geometria desenhada. Neste caso, a diminuição do número de testes leva a que a informação de visibilidade esteja menos actualizada o que afecta a capacidade do algoritmo de remover geometria que na realidade se encontra oclusa. A Figura 46 apresenta os resultados das simulações realizadas com a **Cena 2** (configuração **C7**) para analisar o efeito da variação do intervalo de tempo aplicado a nós considerados visíveis. Aos nós oclusos o teste é aplicado todas as *frames*. O melhor desempenho verifica-se na simulação em que foi usado um intervalo de 240 ms.

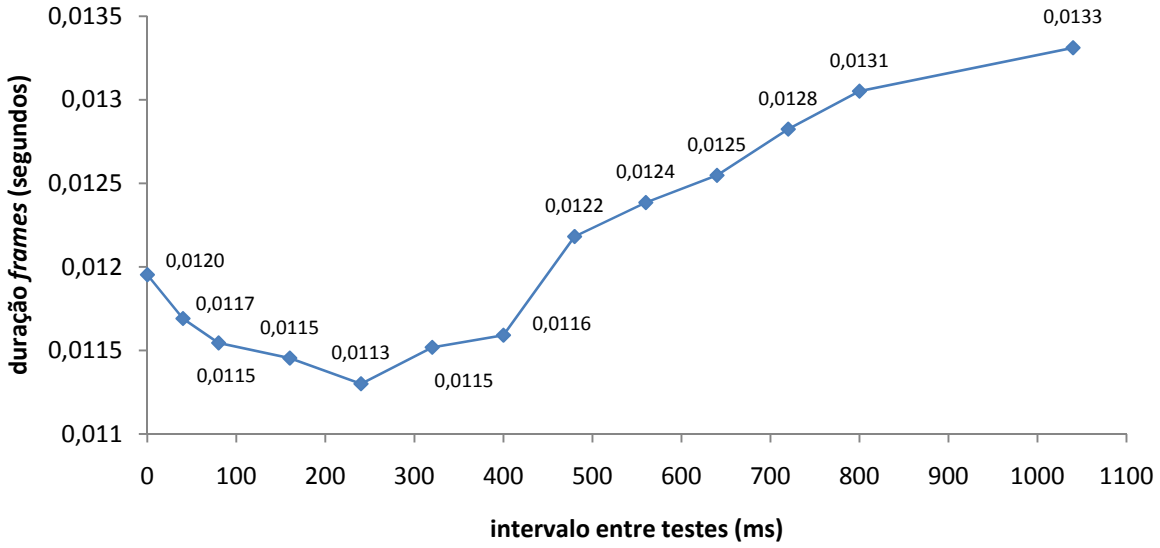


Figura 46 - Duração média de cálculo das *frames* para vários intervalos entre testes a nós visíveis.

Esta forma de optimizar o uso de testes a nós visíveis constitui uma abordagem conservadora. Para além da geometria visível, pode ser enviada para o *hardware* gráfico geometria oclusa. O teste dos nós olcusos em todas as *frames* faz com que a imagem final seja mais correcta ao evitar a perda de objectos visíveis que se verifica com a aplicação do algoritmo de remoção por oclusão original do OSG.

O algoritmo de remoção por oclusão alternativo ao do OSG proposto na Secção 6.2.3 também aborda o problema da realização de testes de oclusão por *hardware* a nós visíveis. Este algoritmo inclui algumas das ideias propostas em [KOV05] e [MAT08]. Em alternativa a realizar o teste a nós visíveis com um intervalo fixo, este método permite variar o intervalo de tempo usado com base no historial de visibilidade dos nós. Desta forma, nós que são sucessivamente determinados como visíveis são sujeitos a menos testes de visibilidade com o decorrer da simulação. O teste é aplicado em todas as *frames* aos nós oclusos. A função *novo_intervalo_de_tempo* é usada no algoritmo para actualizar o intervalo de tempo entre testes, com base no número de testes de oclusão por *hardware* consecutivos que classificaram um nó como visível. Além deste valor, a função também necessita de um parâmetro (*TEMPO_BASE*) que estabelece a gama de intervalos de tempo a usar. Este parâmetro deve ser ajustado de acordo com o tipo de cena e velocidade do ponto de vista.

Com base nos resultados apresentados na Figura 46, o intervalo de tempo entre testes a nós visíveis que oferece melhor desempenho é o de 240 ms. Nestas experiências, o *TEMPO_BASE* foi seleccionado de forma a que o intervalo seja ajustado em torno deste valor (entre 0 e aproximadamente 2x240 ms).

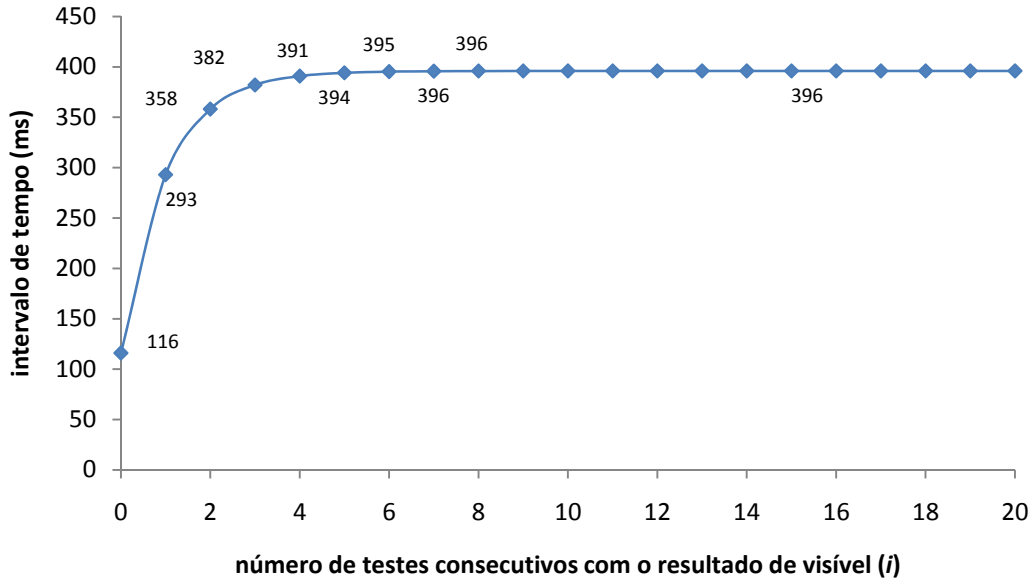


Figura 47 – Curva obtida através da função *novo_intervalo_de_tempo* para um *TEMPO_BASE* de 400 ms.

Usou-se o valor de 400 ms. Através da Figura 47 é possível verificar que o intervalo de tempo de 240 ms é atingido logo após o nó ser considerado visível. Este valor é o que permite o melhor desempenho visto que na simulação onde é percorrido o trajecto na **Cena 2** (configuração **C7**), muitos edifícios transitam entre estados de visibilidade frequentemente. Assim, a probabilidade de um nó rapidamente mudar o seu estado de visibilidade é elevada o que leva a que a sua visibilidade deva ser testada com frequência. Os espaços existentes entre os edifícios permitem que muitos objectos sejam visíveis mesmo que por uma pequena porção, apesar da densidade de geometria. Neste caso, uma pequena alteração no ponto de vista pode alterar o estado de visibilidade de um grande número de objectos. A Figura 48 apresenta os resultados de diversas simulações, onde o intervalo de 400ms é o que proporciona o melhor desempenho.

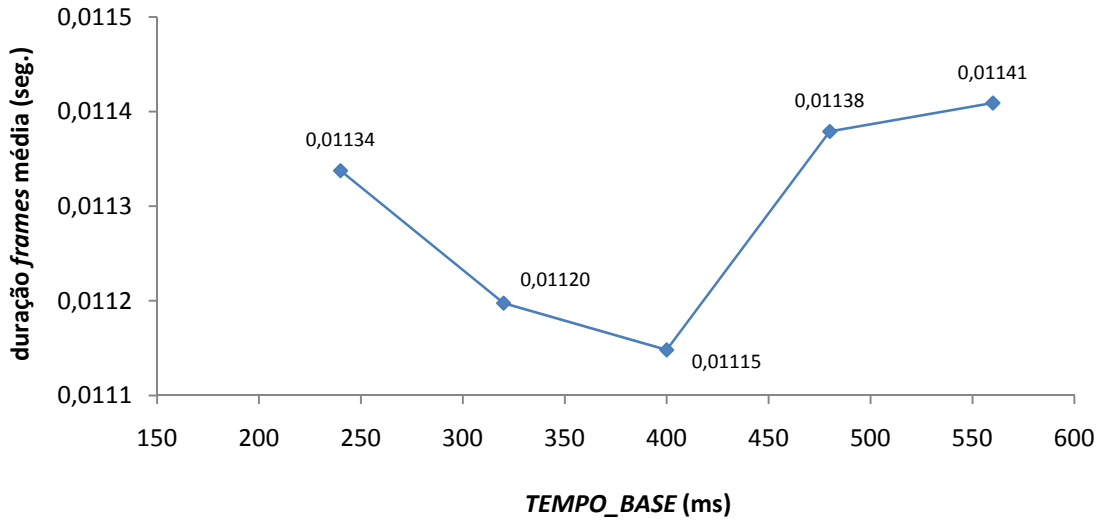


Figura 48 - Simulações realizadas com diferentes intervalos de tempo entre testes aos nós visíveis.

A Tabela 3 compara alguns dados referentes aos vários métodos de remoção de geometria usados ao longo do presente capítulo. As colunas da tabela referem-se, da esquerda para a direita, à aplicação apenas do *view-frustum culling*, aplicação do princípio da coerência temporal a nós visíveis e oclusos (com intervalo fixo entre testes), à remoção por oclusão com intervalo fixo a nós visíveis e em todas as *frames* a nós oclusos e, por fim, ao algoritmo alternativo que usa um intervalo de tempo variável entre testes aos nós visíveis. Todos os valores são relativos a simulações com a **Cena 2** com a configuração de nós OQ **C7**.

Tabela 3 - Comparação de estatísticas obtidas com diferentes métodos de remoção de geometria.

	Apenas *view-frustum culling*	OSG original, intervalo fixo a todos os nós	Intervalo fixo a nós visíveis (240 ms)	Intervalo variável a nós visíveis (400 ms)
Tempo médio por *frame* (segundos)	0,1361	0,0096	0,0113	0,0111
Média de vértices desenhados por *frame*	44 679 143,7	449 119,5	435 496,5	435 644
Número médio de testes de oclusão por *hardware* realizados por *frame*	-	134,4	341,2	329,2

Enquanto a abordagem original do OSG privilegia a velocidade, a realização de testes de oclusão por *hardware* em todas as *frames* a nós oclusos permite que a imagem final seja correcta. Os dois métodos apresentados nesta secção, alternativos ao algoritmo de remoção por oclusão original do OSG, exibem um desempenho bastante semelhante. Ambos conseguem uma redução no número de vértices desenhados à custa do aumento do número de testes de oclusão por *hardware* sem que o tempo por *frame* sofra significativamente em relação o método do OSG. O problema das oscilações que se verifica nas curvas das repostas das simulações que usam um intervalo entre nós oclusos também é eliminado com os dois métodos apresentados nesta secção.

A Figura 49 apresenta as respostas de duas simulações com a **Cena 2** (**C7**). Com o algoritmo alternativo que usa o intervalo variável nos testes a nós visíveis, as oscilações de alta frequência diminuem significativamente devido à geometria oclusa ser continuamente testada. Assim, a inclusão/remoção intervalada de grandes quantidades de geometria é evitada. Outro facto a salientar é o custo associado à realização dos testes de oclusão por *hardware*. O método original, apesar de desenhar mais vértices, tem um tempo por *frame* ligeiramente inferior devido ao menor número de testes que realiza por *frame* e aos objectos que não são desenhados apesar de visíveis.

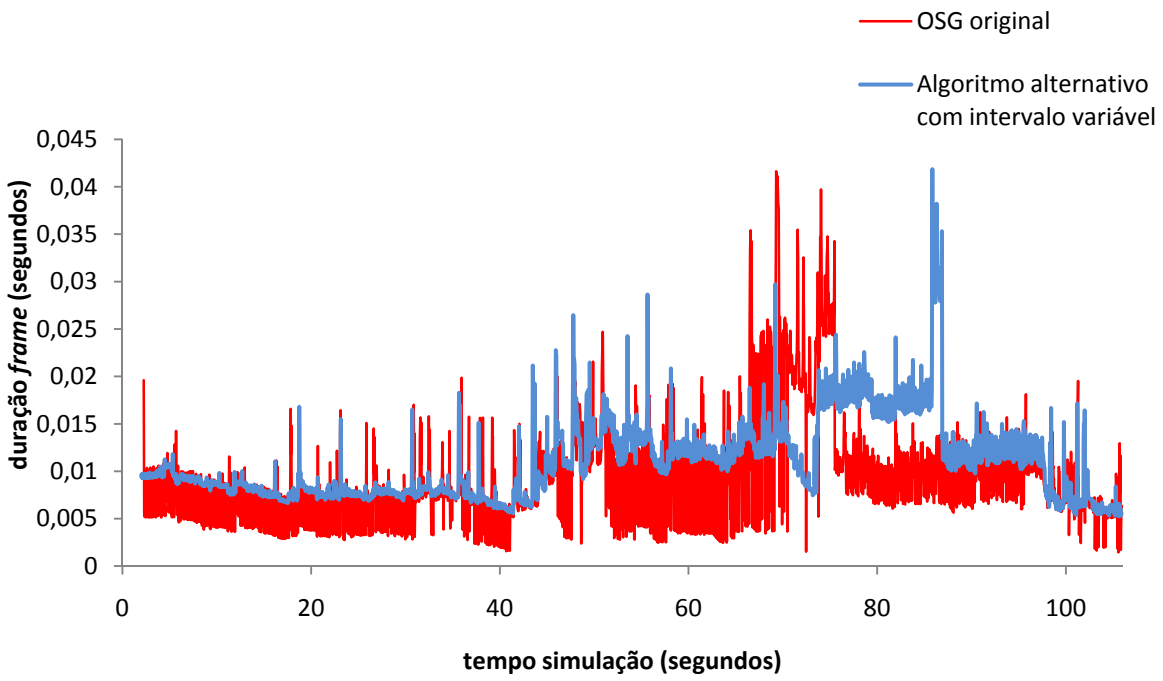


Figura 49 - Diferença resultante de aplicar os testes de oclusão a nós oclusos em todas as *frames*.

6.7. SUMÁRIO

No início do capítulo são apresentadas várias directivas a ter em consideração na aplicação dos testes de oclusão por *hardware* a estruturas de organização de dados hierárquicas. A influência da remoção hierárquica de geometria por oclusão assistida por *hardware* foi comprovada num conjunto de simulações realizadas com duas cenas com densidades de geometria distintas criadas para o efeito.

Em ambientes virtuais com baixa densidade de objectos, a aplicação da remoção por oclusão nunca provou ser consistentemente melhor que o simples *view-frustum culling*. O espaçamento entre oclusores não permite que os volumes envolventes de nós que agrupam vários objectos se tornem oclusos, o que permitiria remover grandes quantidades de geometria com um único teste. Quando aplicados ao nível da geometria, e pelas mesmas razões, os testes determinam maioritariamente a geometria como visível. Esta facto apenas adiciona custos ao cálculo de cada *frame* sem que haja benefício devido a geometria removida por oclusão.

Em cenas com elevada densidade de geometria a mesma situação não se verifica. Neste caso, verificarem-se ganhos significativos no desempenho com a aplicação dos testes de oclusão por *hardware* em todos os níveis hierárquicos do grafo de cena. A densidade de objectos permite uma boa oclusão, o que proporciona que volumes envolventes de grande dimensão possam ser determinados como oclusos. Esta situação permite que, de forma hierárquica, se evite o envio para o *hardware* gráfico de muita geometria com a aplicação de um único teste de oclusão.

A aplicação do princípio da coerência temporal demonstrou permitir bons resultados no tipo de cenas usadas. A aplicação deste princípio poder ser mais problemático em cenas dinâmicas ou em cenas em que o ponto de vista se mova com grande velocidade. Nestes casos, o período de tempo que o resultado de um teste de oclusão por *hardware* pode ser considerado válido varia com a natureza da cena em questão, o que torna difícil generalizar estes valores. O método de remoção por oclusão proposto, alternativo ao do OSG, demonstra um desempenho temporal semelhante a este mas evita as perdas de objectos que ocorrem devido à aplicação da coerência temporal a nós oclusos que diminuem a qualidade de imagem final apresentada.

7. CONCLUSÕES

Esta dissertação apresenta uma introdução aos aspectos relacionados com a determinação de visibilidade de objectos usados em cenas 3D, em particular, a detecção e remoção de objectos oclusos que não contribuem para a imagem apresentada ao utilizador no decorrer de uma simulação visual. Diferentes algoritmos de remoção de geometria assistida pelo *hardware* gráfico foram apresentados e discutidos. O mecanismo de *hardware* utilizado por estes algoritmos parece ser actualmente a melhor solução para o problema da visibilidade de um objecto. Os testes de oclusão por *hardware* quando aplicados a cenas 3D organizadas numa estrutura de dados hierárquica permitem remover grandes quantidades de geometria oclusa com uma única operação. Para estudar como a aplicação dos testes à estrutura hierárquica influencia o desempenho global da aplicação, foram realizadas várias simulações com diferentes cenas 3D.

Mesmo recorrendo ao algoritmo de remoção por oclusão do OSG, com as limitações identificadas, os ganhos obtidos no desempenho da aplicação são bastante consideráveis especialmente quando comparados com o uso apenas do *view-frustum culling*. A rapidez na geração de uma *frame* quando os testes de oclusão por *hardware* são aplicados de forma não hierárquica (apenas ao nível da geometria) foi em média 10 vezes superior à situação onde apenas é usado o *view-frustum culling*. Com a aplicação dos testes a todos

os níveis hierárquicos, o cálculo das *frames* revelou-se 14 vezes (em média) mais rápido, também quando comparado com o *view-frustum culling*.

Estes resultados evidenciam a importância da aplicação dos testes de oclusão por *hardware* em vários níveis hierárquicos de cenas com diferentes níveis de complexidade. Em cenas de grandes dimensões com elevada densidade geométrica, a vantagem da aplicação dos testes revela-se ao eliminar grandes quantidades de geometria com um único teste, aproveitando a estrutura hierárquica do grafo de cena. Esta exploração da remoção hierárquica de geometria por oclusão em cenas com bom poder oclusor permitiu os ganhos observados nas simulações.

Foi também proposto nesta dissertação um algoritmo de remoção por oclusão que explora a coerência temporal de forma alternativa ao algoritmo original do OSG. Este algoritmo gere o número de testes de oclusão por *hardware* realizados aos nós considerados visíveis e realiza testes em todas as *frames* a nós oclusos. Esta forma de aplicar os testes de oclusão por *hardware*, permitiu obter um desempenho temporal semelhante ao método original do OSG, melhorando significativamente a qualidade da imagem final apresentada. Esta melhoria deve-se à diminuição da perda de objectos que ocorre no algoritmo do OSG.

As conclusões aqui apresentadas são baseadas em grafos de cenas. No entanto, estas podem ser generalizadas para outras estruturas hierárquicas de organização de cenas 3D.

7.1. MODIFICAÇÕES SUGERIDAS AO OPENSCENEGRAPH

No decorrer do desenvolvimento da aplicação necessária às simulações anteriormente realizadas surgiram necessidades específicas que a API do OSG na sua actual versão não satisfazia. Para as colmatar, foram introduzidas as alterações na API descritas em detalhe na Secção 6.2.2. Apesar de simples, as três modificações sugeridas podem ser incluídas no projecto OSG em futuras versões, e permitem:

- Usar um intervalo de tempo entre testes de oclusão por *hardware*;
- Detectar quando um nó OQ entra no *view-frustum* antes do final do intervalo entre testes de oclusão por *hardware*;

- Definir intervalos entre testes de oclusão por *hardware* diferentes para nós oclusos e nós visíveis.

Qualquer uma das modificações sugeridas pode ser incluída independentemente das restantes e incidem maioritariamente nas classes **osg::OcclusionQueryNode** e **osgUtil::CullVisitor**. A implementação actual da API resultante deste trabalho pode ser utilizada por qualquer aplicação OSG.

7.2. TRABALHO FUTURO

Existem várias direcções que podem ser seguidas em termos de trabalho futuro. Apesar de não ter sido o principal objectivo da dissertação desenvolver um algoritmo de remoção por oclusão, o usado pelo OSG pode ser melhorado, por exemplo, incorporando algumas das ideias sugeridas nos artigos analisados e assim beneficiar a enorme comunidade que usa o OSG.

Outra possibilidade é o desenvolvimento de um método de análise do grafo de cena que baseado em determinados critérios (complexidade, dimensão, dispersão, etc.) introduza automaticamente nós OQ numa etapa de pré-processamento. Assim, em *runtime*, apenas objectos ou zonas da cena que beneficiariam com a aplicação dos testes de oclusão por *hardware* seriam testadas. De forma complementar, também pode ser desenvolvido um método que ajuste automaticamente o intervalo entre testes de oclusão por *hardware* mediante as condições da simulação num dado instante.

Para comprovar de forma precisa a eficiência das alterações propostas, está prevista a realização de testes sobre as cenas 3D utilizadas em alguns dos trabalhos descritos no Capítulo 4.

Está ainda prevista a aplicação das ideias de organização da cena resultantes deste trabalho ao simulador de condução *Dris* [LEI97].

Referências Documentais

[AIL00] AILA, Timo - *Umbra: A Visibility Determination Framework for Dynamic Environments*. Helsinki University of Technology, Department of Computer Science, 2000. Tese de Mestrado.

[AIL04] AILA, Timo; MIETTINEN, Vielle - dPVS: An occlusion culling system for massive dynamic environments. *IEEE Computer Graphics and Applications*, vol. 24, nº 2, p. 86–97, Março 2004. ISSN 0272-1716.

[AKE08] AKENINE-MÖLLER, Tomas; HAINES, Eric; HOFFMAN, Naty - *Real-Time Rendering*. 3ª ed. A K Peters Ltd, p. 658-680, 2008. ISBN 978-1-56881-424-7.

[ARB03] ARB (Architecture Review Board) OpenGL extension specification – *GL_ARB_occlusion_query*. 2003. http://www.opengl.org/registry/specs/ARB/occlusion_query.txt

[BAR98] BARTZ, Dirk; MEIßNER, Michael; HÜTTNER, Tobias - Extending Graphics Hardware for Occlusion Queries in OpenGL. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware*. Portugal: Lisboa, p. 97–104, 1998. ISBN 0-89791-097-X.

[BAR05] BARTZ, Dirk; KLOSOWSKI, James T.; STANEKER, Dirk - *Tighter Bounding Volumes for Better Occlusion Culling Performance*. Wilhelm Schickard Institute for Computer Science (WSI/GRIS), University of Tübingen, 2005. Relatório técnico.

[BEN75] BENTLEY, Jon Louis - Multidimensional binary search trees used for associative searching. *Communications of the ACM*, vol 18, nº 9, p. 509-517, 1975. ISSN 0001-0782.

[BIT03] BITTNER, Jiri; WONKA, Peter - Visibility in computer graphics*. Jornal Environment and Planning B: Planning and Design*, vol. 5, nº 30, p. 729–756, Setembro 2003. ISSN 0265-8135.

[BIT04] BITTNER, Jiri; WIMMER, Michael; PIRINGER, Harald; PURGATHOFER, Werner - Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum (Proceedings of Eurographics 2004)*, vol. 23, nº 3, p. 615-624, Setembro 2004.

[CAT74] CATMULL, Edwin Earl - *A Subdivision Algorithm for Computer Display of Curved Surfaces*. University of Utah, Department of Computer Science, 1974. Tese de doutoramento.

[COH03] COHEN-OR, Daniel; CHRYSANTHOU, Yiorgos; SILVA, Cláudio T.; DURAND, Frédo - A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, nº 3, p. 412-431, 2003. ISSN 1077-2626.

[COR02] CORRÊA, Wagner T.; KLOSOWSKI, James T.; SILVA, Cláudio T. - Fast and Simple Occlusion Culling. In *Game Programming Gems 3*. Charles River Media, 2002. ISBN 978-1-58450-233-3.

[D3D] *Direct3D 10 Graphics*. Microsoft Developer Network. 2009. http://msdn.microsoft.com/en-us/library/bb205066%28VS.85%29.aspx

[GAM95] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John M. - *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995. ISBN 978-0-201-63361-0.

[GIG88] GIGUS, Ziv; CANNY, John F.; SEIDEL, Raimund - *Efficiently Computing and Representing Aspect Graphs of Polyhedral Objects*. Computer Science Division, University of California at Berkeley, 1988. Relatório técnico.

[GLA84] GLASSNER, Andrew S. - Space Subdivision for Fast Ray Tracing, *IEEE Computer Graphics & Applications*, vol. 4, nº 10, p. 15-22, Outubro 1984.

[GRE93] GREENE, Ned; KASS, Michael; MILLER, Gavin - Hierarchical Z-Buffer visibility. *Proceedings of SIGGRAPH 93*, p. 231–238, 1993. ISBN 0-89791-601-8

[GRO95] GRÖLLER, Eduard; PURGATHOFER, Werner - *Coherence in computer graphics*. Institute of Computer Graphics and Algorithms, Vienna University of Technology, 1995. Relatório técnico.

[GUT06] GUTHE, Michael; BALÁZS, Ákos; KLEIN, Reinhard - Near optimal hierarchical culling: Performance driven use of hardware occlusion queries. *Proceedings of Eurographics Symposium on Rendering 2006*. The Eurographics Association, Junho 2006.

[HIL02] HILLESLAND, K., SALOMON, B.; LASTRA, A.; MANOCHA, D. - *Fast and Simple Occlusion Culling Using Hardware-Based Depth Queries.* Department of Computer Science, University of North Carolina at Chapel Hill, 2002. Relatório técnico.

[HP97] HP (Hewlett-Packard) OpenGL extension specification – *GL_HP_occlusion_test*. 1997. http://www.opengl.org/registry/specs/HP/occlusion_test.txt

[KLO00] KLOSOWSKI, James T.; SILVA, Cláudio T. - The Prioritized-Layered Projection Algorithm for Visible Set Estimation. *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, nº 2, p. 108-123, Abril 2000. ISSN 1077-2626.

[KLO01] KLOSOWSKI, James T.; SILVA, Cláudio T. - Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, nº 4, p. 365–379, Outubro 2001. ISSN 1077-2626.

[KOV05] KOVALCÍK, Vít; SOCHOR, Jirí - Occlusion culling with statistically optimized occlusion queries. *Proceedings of WSCG (Short Papers)*, p. 109–112, 2005.

[KUM96] KUMAR, Subodh; MANOCHA, Dinesh; GARRETT, William; LIN, Ming - Hierarchical Back-face Computation. *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. Portugal: Porto, p. 235–244, 1996. ISBN 3-211-82883-4.

[LEI97] LEITÃO, J. M.; Coelho, A.; Ferreira, F.N. – Dris - A virtual Driving Simulator. *Proceedings of the Second International Seminar on Human Factors in Road Traffic*, 1997. ISBN 97-2-8098-25-1.

[MAR07] MARTZ, Paul - *OpenSceneGraph Quick Start Guide*. Skew Matrix Software LLC, 2007.
http://www.lulu.com/content/767629

[MAT08] MATTAUSCH, Oliver; BITTNER, Jirí, WIMMER, Michael - CHC++: Coherent: Hierarchical Culling Revisited. *Computer Graphics Forum (Proceedings Eurographics 2008)*, vol. 27, nº 2, p. 221–230, Abril 2008. ISSN 0167-7055.

[MEI99] MEIßNER, Michael; BARTZ, Dirk; HÜTTNER, Tobias, MÜLLER, Gordon; EINIGHAMMER, Jens - *Generation of Subdivision Hierarchies for Efficient Occlusion Culling of Large Polygonal Models.* Wilhelm Schickard Institute for Computer Science (WSI/GRIS), University of Tübingen, 1999.
Relatório técnico.

[NV01] NVIDIA Corporation OpenGL extension specification – *GL_NV_occlusion_query*. 2001.
http://www.opengl.org/registry/specs/NV/occlusion_query.txt

[OSG] OpenSceneGraph. 28/09/2009.
http://www.openscenegraph.org

[OpSG] OpenSG - Open Source Scenegraph. 28/09/2009.
http://www.opensg.org

[PAB02] PABST, Thomas - *ATi Takes Over 3D Technology Leadership With Radeon 9700: HyperZ III*. 2002.
http://www.tomshardware.com/reviews/ati-takes-3d-technology-leadership-radeon-9700,491-9.html

[ROH94] ROHLF, John; HELMAN, James - IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. *Proceedings of ACM SIGGRAPH*, p. 381–394, 1994. ISBN 0-89791-667-0.

[RUB80] RUBIN, Steven M.; WHITTED, Turner - A 3-Dimensional Representation for Fast Rendering of Complex Scenes. *Proceedings of ACM SIGGRAPH 1980*, p. 110-116, 1980. ISBN 0-89791-021-4.

[SCO98] SCOTT, Noel D.; OLSEN, Daniel M.; GANNETT, Ethan W. - An overview of the VISUALIZE fx graphics accelerator *hardware*. *Hewlett-Packard Journal*. Vol. 49, nº 2, p. 28–34, Maio 1998.

[SGI] SGI Developer Central Open Source. *OpenGL Performer*. 28/09/2009. http://oss.sgi.com/projects/performer/

[STA04] STANEKER, Dirk; BARTZ, Dirk; STRAßER, Wolfgang - Occlusion Culling in OpenSG PLUS. *Computers & Graphics*, vol. 28, nº 1, p. 87–92, Fevereiro 2004.

[STL99] *Standard Template Library Programmer's Guide*. Silicon Graphics Computer Systems, Inc. 1999. http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi/0650/bks/SGI_Developer/STL_PG

[TEL91] TELLER, Seth J.; SÉQUIN, Carlo H. - Visibility Preprocessing for Interactive Walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91)*, vol. 25, nº 4, p. 61–70, Julho 1991. ISSN 0097-8930.

[WER94] WERNECKE, Josie - *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor*. Addison-Wesley Professional, 1994. ISBN 0-201-62495-8.

[WON00] WONKA, Peter; WIMMER, Michael; SCHMALSTIEG, Dieter - Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs. *Rendering Techniques 2000 (Proceedings of the Eurographics Workshop on Rendering 2000)*, p. 71–82, 2000. ISBN 3-211-83535-0.

[WOO97] WOO, Mason; NEIDER, Jackie; DAVIS, Tom - *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. 2ª ed. Addison–Wesley, 1997. ISBN 978-0201461381.

[ZHA97] ZHANG, Hansong; MANOCHA, Dinesh; HUDSON, Tom; HOFF III, Kenneth E. - Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH 97*, p. 77–88, 1997. ISBN 0-89791-896-7.